A new Content Distribution Network architecture - PlentyCast

CAO WEI QIU

KTH Microelectronics
and Information Technology

Master of Science Thesis
Stockholm, Sweden 2004

IMIT/LCN 2004-05

A new Content Distribution Network architecture - PlentyCast

by
Cao Wei Qiu
KTH/IMIT and SCINT
mike@scint.org or kthmike@kth.se

2004-April-30
Stockholm, Sweden
A thesis presented to the Royal Institute of Technology, Stockholm
in partial fulfillment of the requirement for the degree of
Master of Science in Internetworking

Academic Advisor and Examiner:
G. Q. Maguire Jr.
Department of Microelectronics and Information Technology (IMIT)
Royal Institute of Technology (KTH)

Industry Supervisor:
Lars-Erik Eriksson
Swedish Center of Internet Technology (SCINT)

Signature: _____ **Signature:** _____

Date: _____ **Date:** _____

Abstract in English
Content Distribution Networks have existed for some years. They involve the following problem domains and have attracted attention both in academic research and industry: content replica placement, content location and routing, swarm intelligence, and overlay network self-organization for this type of distributed system. In this project, we propose a novel Content Distribution Network architecture – PlentyCast. This study focuses on improving access latency, network scalability, high content availability, low bandwidth consumption, and improving infrastructure performance for Content Distribution Networks. Outstanding problems such as: Flash crowd, DoS, and difficulty of traffic engineering due to Peer-to-Peer are addressed.

Abstract in Swedish
Mediadistributionsnätverk har funnits några år. De har fått uppmärksamhet i både akademisk forskning och i industrin och kännetecknas av följande frågor: placering av innehållskopior, lokalisering av innehåll och routing, svärm intelligens, överlagrade nätverks självorganisering för denna typ av fördelade system. I denna rapport studeras en ny nätverksarkitektur för innehållsfördelning - PlentyCast. Denna studie fokuserar på tillgångslatens, nätverksskalbarhet, hög innehållstillgång, låg bandbreddskonsumtion, och förbättrad infrastrukturprestanda för Innehållsfördelningsnätverk.

Acknowledgement
Thanks to Mr. Bengt Källbäck's for his recommendation of this job in SCINT. Thanks to Mr. Lars-Erik Eriksson in SCINT to select me for this project. He greatly supports my work by taking time to content distribution discuss with me. Thanks to David Jonsson from Interactive Institute for his input of the idea of using Peer-to-Peer techniques. Thanks to my teacher Professor Vlad Vlassov for many discussions. Thanks to my colleagues in SCINT, Kjell Torkelsson, Eriksson B. Svante, Lennart Helleberg, and Staffan Dahlberg gave me their warm-heart assistance during my work at SCINT.

My parents' encourage from China gives me great deal of confidence, enabling me to have solve every problem regarding living and working in Sweden. Thanks for their deep love and support in my life! My Godparents – Eva Lodén and Ragnar Lodén have their greatly helped in my life and work during my days in Sweden. Their love made it much easier live and work in Sweden. Thanks to Per Pedersen in Ericsson for his strong recommendation to the Royal Institute of Technology.

My dream is to work as a teacher in either academia or as a coach in industry. This is because Professor Gerald Q. Maguire Jr.- my academic advisor and examiner, became my model for this dream. I like his high expectations for my examination and his very helpful advice role. This model impressed me and awoke my enthusiasm for teaching or coaching. Thank you, Chip!

A well-known principle has been demonstrated once again: any achievement is not due just to one person, but due to many people.

Table of Content

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction to Content Distribution Network

In this Chapter, I will give an introduction of Content Distribution Networks technology. This includes examining three aspects for each of them: (1) problems to be resolved in this realm, (2) core techniques have been used to support to this approach, and (3) hot issues related in each realm.

The following chapters have been arranged in this way: second chapter will introduce Peer-to-Peer technology, third chapter will have an introduction of swarming content delivery and Forward Error Correction techniques. A compact introduction of Mobile Agent technology will be conducted in chapter four. After all technologies which we are interested in this project, a problem statement will be made to elaborate each goal that we set up in chapter 5. I will explain our motivation and understanding towards each problem which we are interested in solving, problem model, and research criteria in this project. In chapter 6, we our proposal of a highly usable, scalable, and reliable Content Distribution architecture. At the end of this chapter, I will conduct case studies to evaluate if PlentyCast fulfills the project goals. Finally we will conclude our work and highlight our future work.

1.1 Content distribution over the Internet

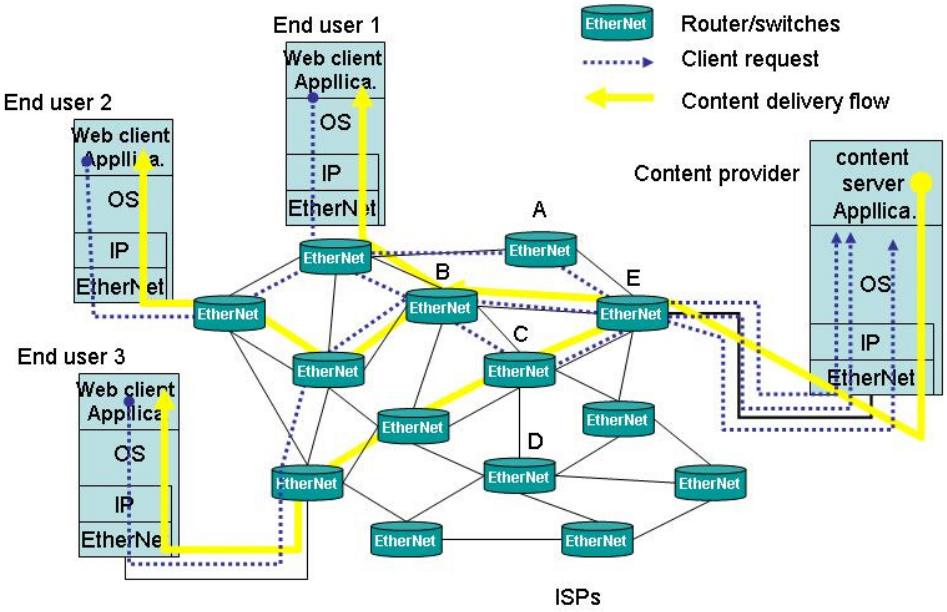


Figure 1. An overview of how content is distributed or delivered to its users

When the Internet bubble was breaking in 1998, many people realized that publishing on a web site is only one step of hosting a web site, and the most important goal is to get the web content delivered to the users over the network. Figure 1 shows an overview of how content will be distributed or delivered to the users. Here, a client first sends a request to a content server via an application layer protocol such as HTTP [38]. After the request has been accepted by the content server, the server sends the content to the client over the network links across different routers and or switches. From a hardware perspective, both client and server are similar; and they both are likely to be connected to the Internet via an Ethernet Network Interface Card. The

major difference between client and server is related to how their software[1] is structured. The Internet connects the users and the content providers.

In Figure 1, there are three actors: the user, the content provider, and the ISPs who provide the network between the user and the content provider. They each have the different requirements based on their own role. From a user's perspective, the expectation is fast access to the content they want at any time they want it. In addition, the user expect to have good quality of the delivered content. From content provider's perspective, the expectation is that their content should be maximally available for all the users who want to access, this should be only limited by the performance of their content server. From the ISPs' perspectives, they expect to have larger number of users utilizing their access network, while minimizing the bandwidth consumption on interconnections to their networks; expect high performance from their network infrastructure.

1.2 Internet structure

By definition, Internet is a network of networks. The Internet is a well-known example, being made up of thousands of different networks (also called Autonomous Systems or AS's) that communicate by using the IP protocol (see Figure 2). These networks range from large backbone providers such as UUNet and BBN to small local ISPs such as Swipnet in Stockholm's Solna. Each of these networks is a complex entity in itself, physically being made up of routers, switches, fiber, microwave, ATM, Ethernet, etc. All of these components work together to transport packets through the network toward their destinations. In order for the Internet to function as a single global network interconnecting everyone, all of these individual networks must connect to each other and exchange traffic. This happens through a process called peering. When two networks decide to connect and exchange traffic, a connection called a peering session is established between a pair of routers, each located at the border of one of the two networks. These two routers periodically exchange routing information, thus informing each other of the destinations that are reachable through their respective networks. There exist thousands of peering points on the Internet, each falling into one of two categories: public or private. Public peering occurs at major Internet interconnection points such as MAE-East, MAE-West, and the Ameritech NAP, while private peering arrangements bypass these points. Peering can either be free such as the one between Tier-1 ISPs, or one network may purchase a connection to another such as Tier-3 and Tier-2 ISP to Tier-1 ISPs. Once the networks are interconnected at peering points, the routing protocol running on every Internet router moves packets in such a way as to transport each data packet to its correct destination. For scalability purposes, there are two types of routing protocols directing traffic on the Internet today. IGP (Interior gateway protocols) such as OSPF and RIP create routing paths *within* individual networks or ASs, while the EGP (exterior gateway protocol) BGP (Border Gateway Protocol) is used to send traffic between different networks. Interior gateway protocols often use detailed information about network topology, bandwidth, and link delays to compute routes through a network for the incoming packets. Since this approach does not scale up to handle a large-scale networks composed of separate administrative domains, BGP is used to link individual networks together to form the Internet. BGP creates routing paths by

[1] Definition of client-server http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?client-server accessed on 2004-01-19 22:45

simply minimizing the number of individual networks (called Autonomous Systems) a packet must traverse. While this approach does not guarantee that the routes are even close to optimal, it supports a global Internet by scaling to handle thousands of ASs and allowing each of them to implement their own independent routing policies within the AS. Peering points and routing protocols thus connect the disparate networks of the Internet into one cooperating infrastructure. Connecting to one of these networks automatically provides access to all Internet users and servers. This structure of the Internet as an interconnection of individual networks is the key to its scalability. It enables distributed administration and control of all aspects of the Internet system: routing, addressing, and internetworking. However, inherent in this architecture are four types of bottlenecks that, left unaddressed, can slow down performance and decrease the ability of the Internet to handle an exponentially growing number of users, services, and traffic. These bottlenecks are described in the following sections.

1.3 Internet bottlenecks

Here bottleneck refers to network performance bottleneck[2]. This occurs when the desired data transmission rate between two end systems exceeds available link capacity along the path for a certain period of time for a given topology. Consequently it degrades network performance by increasing packet loss rate, increasing end-to-end latency, and introducing jitter[3]. However, as we only consider static content, thus jitter is irrelevant. These problems can be divided into four classes: first-mile, backbone, peering, and last –mile. The following figure depicts an overview of these problems.

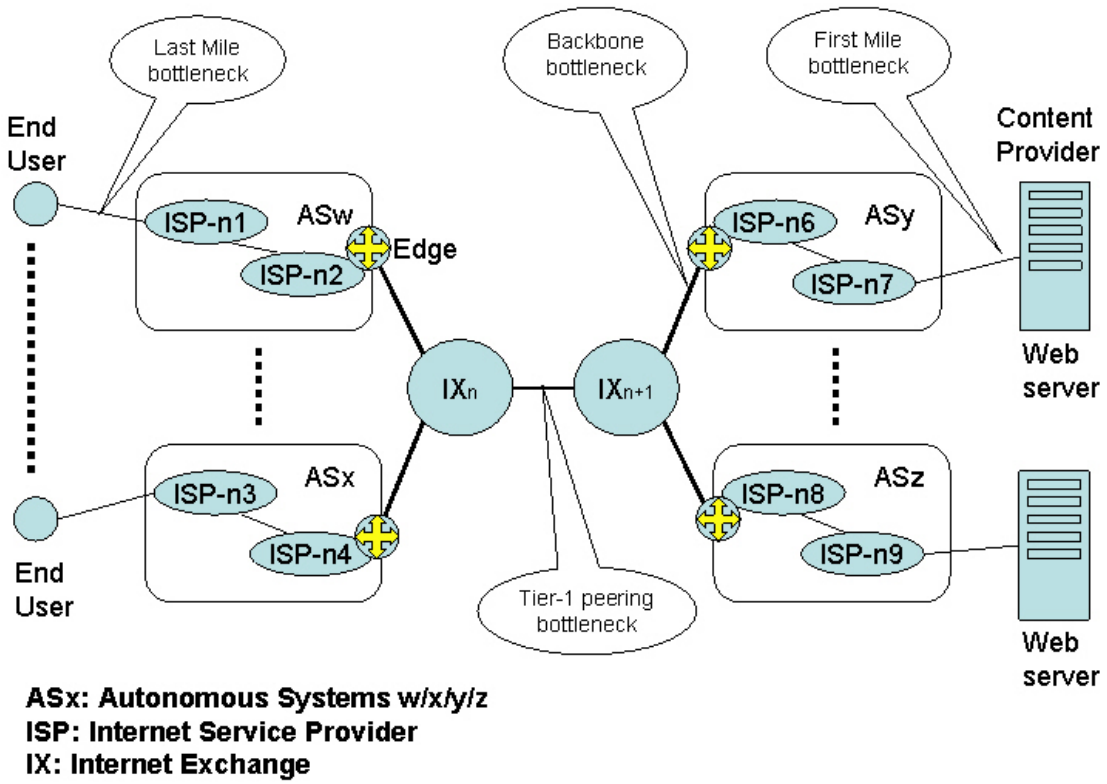


Figure 2 Four classes of bottlenecks on today's Internet

[2] http://en.wikipedia.org/wiki/Performance_problem Accessed on 2004-01-11 10:25
[3] http://en.wikipedia.org/wiki/Jitter accessed on 2004-01-11 10:53

1.3.1 First-mile bottleneck

The first-mile problem appears when link capacity between local ISP and the content server limits the number of users who wants to access the content server. Intuitively, the capacity of the link between local ISP and its customer – the content server, is a constant for a certain typical long period of time. The number of arriving user requests is actually a random distribution because it is a stochastic process. If there are a small number of clients accessing a content server, the total desired access rate will be less than the link capacity. In this case, the packet loss rate, and latency will be acceptable. But when the content server becomes a hot spot, viz. when large numbers of clients access the same content server, the total desired access rate will exceed the maximal bandwidth that the link can provide. In this case, high packet loss rate will result from the link congestion unless the ISP can replicate responses for common requests. Second, high latency can cause very long response times for user requests. Thirdly, the high traffic load can overwhelm the CPU or memory resources of the content server, ultimately bring down this server. Together they are the first mile problem. One solution is to increase the link capacity, but this is not an optimal solution because this will lead to potential bandwidth wasting when the peak hours passed due to the stochastic process of client requests and it is not a cost-effective solution from the content provider's perspective. Since CDNs replicate or cache the content in their replica servers or cache proxies distributed across many different ASs, this link budget of the first mile has been distributed to many limited links between the access clients and the content replicas (replica servers or proxies). In this way, the bandwidth between the local ISP and the hot spot content server has been saved to be a great extent. In addition, this relieves the original content server from potential overload. Link budgets of a content provider can be reduced to a small number because each CDN only needs a few links between the content server and the replica servers for content updating because the CDNs can distribute/create replicas in their own overlay networks.

1.3.2 Peering bottleneck problem

The peering bottleneck occurs due to two major reasons. Firstly, lower tier ISPs rent their upstream links from higher tier ISPs. But for ISPs within the same tier, they do not constantly pay each other for the links to connect to each other. This leads to a peering problem – these links often run at a fixed capacity for a long time. Secondly, installation of new circuits to expand the capacity of these links occurs much slower than the actual traffic demands increase due to many technical or non-technical reasons. Imagine that thousands of users only need a mouse click but actually installing new links takes at least 3-5 months. Therefore, many ISPs don't wait until the link is saturated before expanding capacity becomes an issue. For example, Telia[4] starts to expand its peering links when traffic reaches 50% of capacity. This seems to make the peering bottlenecks less serious than the other bottlenecks. However, increasing with numbers of broadband Internet users, bandwidth intensive applications, and slow upgrade of links bottlenecks in the peering groups among tier-1 ISPs can occur. This class of bottlenecks we call peering bottlenecks.

[4] www.telia.se accessed on 2004-01-18 19:21. Telia is one the largest ISPs in Europe and North American.

1.3.3 Backbone bottleneck

For the same reasons, the backbone of the Internet is under increasing pressure from traffic demands. However, the speed of installation of new capacity is even slower than the peer link expansion. Thus traffic engineering is used to maintain reasonable Quality of Service for upper layer services. However, it is hard to shape the traffic in today's increasingly diversified services due to the complexity of trade-offs amongst upper tier ISPs and local access users and their limited bandwidth. The difficulties of doing traffic engineering and new capacity expansion decision may cause the backbone bottlenecks on the Internet.

1.3.4 Last mile bottleneck

If the end user's bandwidth intensive applications run over a low bandwidth dialup or cable-modem link to their local ISP, this may become a bottleneck on the last-mile link. This class of bottlenecks exists for the connection between end users and their local ISP. Increasingly, the last-mile problem seems to be resolved by Digital Subscriber Line access which quickly rolled out over the world [13]. However, it actually only relaxes the bandwidth constraint between the user's modem and the xDSL distribution point (a DSLAM[5]). Unfortunately, the primary link from the ISP's core switches to the distribution cabinet often becomes another bottleneck. Furthermore, this will potentially cause peering and infrastructure bottlenecks into more serious problem. Even though this backhaul link is easier to expand than any other bottleneck links and the number of the subscribers attached to this link is much easier to predict than on other bottlenecks, you do not know what bandwidth intensive applications the end users are like to run. When an ISP does traffic engineering, it must avoid blocking certain bandwidth intensive applications or this could cause the end users to immediately subscribe to a competitor's network. While the xDSL rollout resolves the last-mile problem, it challenges traffic engineering and existing peering and first-mile solutions.

To sum up, the problems that a CDN faces are to meet the requirements of the end user, content provider, and the ISP in the context of these four classes of Internet bottlenecks.

1.4 CDN technologies

1.4.1 A system overview

As we described in the previous sections, a CDN's goal is to (1) minimize the latency, (2) maximize the content data availability, and (3) minimize the bandwidth consumption on ISP networks. Therefore, we can define that a CDN is an overlay network which is used to transfer a large amount of frequently requested data in a short time to clients. More systematically definitions have been given such as: Protocols and appliances created exclusively for the location, download, and usage tracking of content [24]. This means that a CDN provides:

(1) A way to distribute content and applications globally through a network of strategically placed servers;
(2) A way to track, manage, and report on the distribution and delivery of content; and

[5] Digital Subscriber Line Access Multiplexer.

(3) A way to provide better, faster, and more reliable delivery of content and applications to users.

This following figure depicts an overview of a Content Distribution Network.

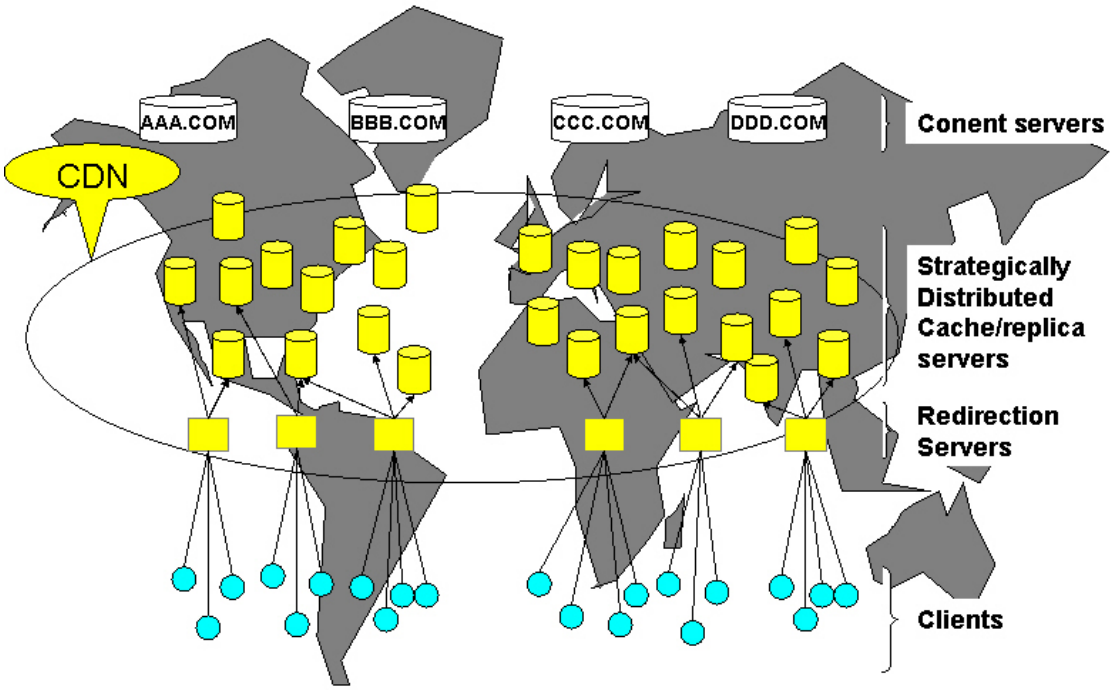


Figure 3 Overview of an typical CDN

This overview shows us that content servers delegate to replica servers copies of the content. When a user accesses certain content delegated via the CDN, they actually access the content not from the original content server, but from a replica server even from multiple replica servers. However, before they get access the content, their requests are redirected by the redirection servers. These servers tell the users to access specific replica servers which are strategically close to clients. The users can now download the content as if the content server was situated in a short distance away (i.e. a smaller numbers of hops compared to accessing the original content server) since it actually download from a nearby replica severs. Thus the latency has been decreased. From another aspects, the content server has less of a danger of being overwhelming when large numbers of clients access the content; since CDN does replication of content across many servers. Furthermore, load balancing causes the client access traffic across to be evenly distributed different replica servers. Thus, the CDN significantly reduces workload at the original content server. In addition, the first mile[6] bottleneck has been alleviated by distributing the traffic load over all the replica servers close to the clients, thus consumption of bandwidth on last mile, peering and even backbone bottlenecks are proportionally reduced. This relaxes the pressure on the ISP for traffic engineering.

1.4.2 A typical architecture

In a typical CDN, the following components are mandatory: client, replica servers, original server, billing and charging systems, request routing system, distribution

[6] Please refer to 2.1.4

system and accounting system. The relationship amongst these components (indicated with numbered lines in Figure 3) is described as follows:

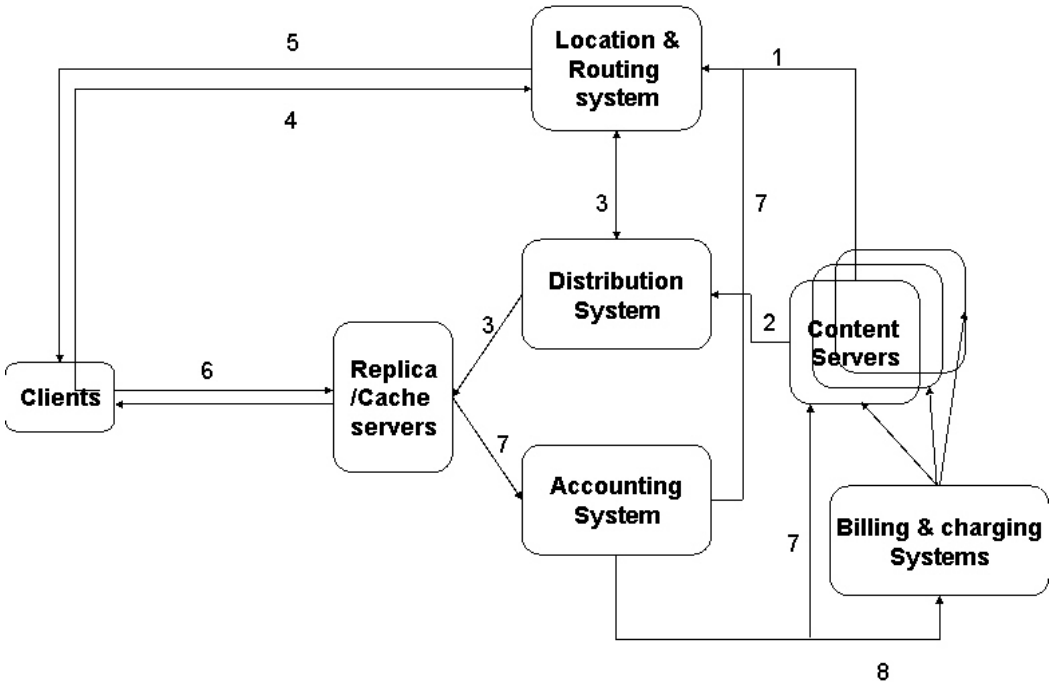


Figure 4. A typical CDN architecture

(1) The original server delegates its Universal Resource Locator name space for objects to be distributed and delivered by the CDN to the request routing system.
(2) The original server publishes content that is to be distributed and delivered by the CDN to distribution system.
(3) The distribution system sends content to the replica servers. In addition, this system interacts with the request routing system through feedback to assist in replica server selection for clients.
(4) The client requests documents from the original server. However, due to URL name space delegation, the client request is redirected to the request routing system (redirection server).
(5) The request routing system routes the request to a suitable replica server.
(6) The selected replica server then delivers the content to the client. Additionally, the replica server sends accounting information for delivered content to the accounting system.
(7) The accounting system collects all accounting information and then manipulates content access records for input to the billing system and statistics are feedback to the request routing system for better redirection of future requests.
(8) The billing system uses the content detailed records to work out how much shall be charged or paid each content providers[2].

Following the above flow, we can have the following description of the components of this CDN system.

A Client is a Hyper Text Transfer Protocol user application. The requirement of user is to be able to access the content at any time when he or she wants.

Replica server is the most important type of component in a CDN. Its major features are: (1) archiving copies of content data strategically considering granularity of the content data; (2) communicating in their peering group in order to achieve load balancing for client traffic; (3) Push content data strategically in according to information distribution system; (4) generating accounting and statistical data.

The location and routing system is mainly responsible for redirecting the client's request from the original content server to specific replica servers. It gets updates from original content servers to determine data granularity whose location shall be redirected, a utilized feedback from accounting system for better redirection based upon certain access metrics, i.e. page hit rate.

The distribution system is the channel to deliver content data to different replica servers. Nowadays, CDNs usually have large numbers of replica servers and they are usually spread widely. How to distribute content data to each replica server and how to manage all the replica servers are the major responsibilities of distribution system. There are at least two popular channels to be used by CDN distribution systems. One is the terrestrial Internet links and the other Satellite links (broadcast is used in this case). Many CDN operators usually choose to construct an overlay network connecting replica server into a tree in order to manage all nodes across the Internet.

The accounting system is to collect all types of statistical data for different uses in other components such as billing and charging system, the location and routing system, and the distribution system.

The content server is the CDN's customer who is willing to pay for distribution services over a Content Distribution Network.

1.4.3 Traditional CDN criteria

In general, we understand how CDN works in such a typical architecture. However, what makes a bold CDN system? To answer this question we examine the desired attributes of a traditional CDN system. In general, they should have the following properties: fast access, robustness, transparency, scalability, efficiency, adaptive, stability, load balancing, interoperability, simplicity [1] and security.

Fast access, from a user perspective, access latency[7], this is the important measurement of CDN usability. A good CDN aims to decreases content access latency. In particular, it should provide the user with lower latency on average, than would be the case without employing a CDN system.

Robustness, from a user perspective, means high content availability, which is another important quality of a CDN. Users expect to receive content whenever they want. From a system design point of view, robustness means that (1) a small number of replica servers or redirection servers might crash, but this should not bring down the entire CDN; and (2) the CDN should recover gracefully in the case of failures. These two attributes actually require good self-organization of the CDN in order to achieve fault resiliency; otherwise the users will see either failed requests of high delay.

[7] Please refer to section 2.1.1 for detailed explanation of latency

Transparency, a CDN system should be transparent for the user, the only thing the user should notice are faster response and higher content availability. This requires that the CDN to be dependent of the user client.

Scalability, since the explosive growth in network size and density in the last decades and continuing exponential growth for the near future, a key to success in such an environment is scalability. A CDN should scale well with both increasing size and density of the Internet. This requires all protocols employed in the caching system to be as lightweight as possible.

Efficiency, from an ISP's point of view, includes two aspects of efficiency. First, how much overhead does the CDN impose on the Internet? The additional load of CDN should be as minimal as possible. This requires that the quantity of signaling packets should be as small as possible. Secondly, any mechanisms of a CDN should avoid leading to critical network resource over-utilization, i.e. increasing pressure on the DNS [47] service.

Adaptive, it's desirable to make a CDN adapt to the dynamically changing user demands and the changing network environment. For instance, a CDN must be able to deal with the flash crowd[136] problem for some special content servers. Adaptation involves several aspects of the system: replica management, request routing, replica server placement, content placement, etc. This increases content availability for the content provider, while load balancing increases robustness.

Stability, from an ISP's point of view, means that a CDN shall not introduce instability into the network. For instance, a naïve CDN routing system distributing requests based upon network information could result in oscillation due to the instability of the Internet. This oscillation will cause the CDN's cost to increase due to content replication and request routing, thus potentially leads to higher latency in delivering content to a user.

Load balancing is desirable, thus the CDN should distribute the load evenly throughout the entire overlay network. It can effectively avoid a single point of failure due to failure of suboptimal replica servers and redirection servers. From the content provider's point of view, this feature alleviates the first mile bottleneck. From an ISP's point of view, this reduces demands for their network bandwidth.

Interoperability is important, since the Internet grows in scale and coverage it spans a wider range of hardware and software architectures. For instance, on the last mile access, xDSL lines connect a vast numbers of households. Additionally, NAT and firewalls are becoming more and more popular. A good CDN must adapt to a wide range of network architectures.

Simplicity is important as simple mechanisms are always easier to correctly implement, and system maintenance is likely to be lower in cost; also simple mechanisms are more likely to be accepted as international standards.

Security is always an important property of today's distributed systems. CDN network security mainly addresses the problems of Digital Right Management of licensed content. Another aspect of security is to secure the CDN network itself. However,

security is often correlated to efficiency, thus optimizing one generally minimize the other. Therefore, the appropriate balance must be found.

1.4.4 Core mechanisms

From the above description, we can see that there are some mechanisms which are essential for CDN systems. They are: server placement, replica placement and management, request routing, and server location. In the following sections, I will try to explain the problems from each of these aspects and describe some state-of-the-art approaches for solving them.

1.4.4.1 Server placement

1.4.4.1.1 Theoretical problem models and solutions

As one of major goal of a CDN, it is very important that a CDN minimize latency between clients and the content server. The problem is where to place servers on the Internet. Intuitively, where to place replica servers (which contains copy of content) is directly in related to average content access latency for clients. Therefore, optimizing server placement must succeed in minimizing access latency. In a traditional CDN, replica server and content replica are coupled. Thus server placement becomes the basis of replica distribution. Theoretically, there are three models used so far to formulate the server placement problem. They are Minimum K-Center problem, the location facility problem [25] and the Minimum K-Center problem and location facility problem with constraints (also quite popularly) [26].

Minimum K-Center problem
Given N servers, select K (K<N) centers (facilities), and then for each location j which is assigned to center i (*i* in N), we shall always have cost $d_j c_{ij}$ (where d_j denotes the demand of the node *j*, c_{ij} denotes the distance between *i* and *j*). The goal is to select the K centers to minimize the sum of these costs.

Location facility problem
Given a set of locations I at which facilities may be built, building a facility at location I incurs a cost of F_i. Each client j must be assigned to one facility, incurring a cost $d_j c_{ij.}$ The objective is to find a solution of the minimum total cost. The difference between this model and K-Center problem is the number of centers. K-Center model place a limitation *i* number of the centers (which is the K), but the Location facility model is open to varying from N.

Limited K-center and location facility problem
In [26], it is denoted as a capacity version problem. In this model, we place more service constraints on both the location facility and the centers. For instance, a mount of services a center can provide and maximal number of requests which a facility can serve can be constraints. This enables the server and replica placement problem to be formulated as either a limited or unlimited location facility problem, or a limited or unlimited minimal K-center problem.

Based upon these problem models, there were many solutions which had been developed. For this NP-hard minimum K center problem, if we are willing to tolerate inaccuracies within a factor of 2, i.e. the maximum distance between a node and the nearest center being no worse than twice the maximum in the optimal case, the

problem is solvable in O (N | E |) time [28] The algorithm can be described briefly as follows:
Given a graph G = (V, E) and all its edges arranged in non-decreasing order by edge cost. c: $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$, let $G_i=(V_i, E_i)$, where $E_i=\{e_1, e_2, \ldots e_i\}$. A square graph of G, G^2 is the graph containing V and edge (U,V) wherever there is a path between u and v in G of at most two hops, $u \neq v$ – some edges in G^2 are pseudo edges, in that they do not exist in G. G = (V, E) is a subset of V' is included in V such that , for all u, v belongs to V' , the edge (u , v) is not in E. An independent set of nodes G^2 thus a set of nodes in G that are at least three hops apart in G. We define a maximal independent set M as an independent set V' such that all nodes in V – V' are at most one hop apart from nodes in V'. The outline of the minimum K-center algorithm from [28] is shown as follows:

1. Construct G_1^2, G_2^2, G_3^2 …, G_m^2
2. Compute M_i for each G_i^2
3. Find smallest i such that $| M_i | \leq K$, for instance j
4. M_j is the set of K center

1.4.4.1.2 Heuristic approaches

Since the theoretical approach is computational expensive or does not consider the network and workload, thus it is difficult to apply in realistic situations [29] and may not be suitable for CDN. Therefore, other heuristic and suboptimal algorithms have been proposed, which consider some practical aspects of the CDN, such as network load, traffic pattern and network topology [26], [29], [30]. They offer (relatively) lowing computational complexity.

After comparing different algorithms such as Tree-based algorithm, Greedy algorithm, Random algorithm, Hot spot algorithm, super-optimal algorithm in simulation, Qiu et al. [26] found that the Greedy algorithm is the one with the best performance, less computational expensive, and relatively insensitive to imperfect data. The basic idea of the greedy algorithm is as follows. Suppose it needs M servers amongst N potential sites. It chooses one site at a time. In the first iteration, it evaluates each of the N potential sites individually to determine its suitability for hosting a server. It computes the cost associated with each site under the assumption that accesses from all clients converge at that site, and picks the site that yields the lowest cost, i.e. lowest bandwidth consumption. In the second iteration, it searches for a second site that, in conjunction with the site which has already been selected, yields the lowest cost. In general, in computing the cost, the algorithm assumes that clients direct their accesses to the nearest server, i.e. one that can be reached with the lowest cost. The iteration continues until M servers have been chosen. To support this greedy approach, one usual method is to partition the graph into tree. K hierarchically Well-Spread Tree[27] (K-HST) is one of the typical representations. However, greedy placement requires knowledge of the client locations in the network and all pairwise intern-node distances. This information in many cases may not be available, for instance, use of NAT and Firewalls might prevent the location of clients.

In [29], a topology-informed placement strategy has been proposed. Assuming that nodes with highest outdegree[8] can reach more nodes with smaller latency, we place

[8] In a directed graph, we say that a vertex has outdegree *x* if there are (exactly) *x* edges leaving that vertex.

servers on candidate hosts in descending order of outdegrees. These are called Transit Nodes due to the assumption that nodes in the core of the Internet transit points will have the highest outdegrees. In most of cases, Autonomous System gateways will be chosen as the transit nodes. However, due to inaccuracy of AS topology information, the authors [29] exploited router level topology information as showed that result is better performance than simply using AS level routing information. Going deeper into the network, they found that each LAN associated with a router is a potential site to place a server, rather than being each AS being a site.

To sum up, considering the most up-to-date solutions for the server placement: greedy and topology-informed placement strategies are both well developed. The edge computing [9] are inspirited these solutions. However, to our best knowledge, the approaches to the real world isn't well exploited.

1.4.4.2 Replica placement

Similar to server placement, replica placement is also a facility location problem, but the difference from server placement is greater concern about user access patterns. The first problem model was formulated in [32]. It considers distance, cache size, and access frequency. As a distance metric, they choose hierarchical distance model to calculate the distance metrics in order to get better approximation to the actual Internetwork.

1.4.4.2.1 A typical cost model

In [33], the another have developed a cost model for object placement over the Internet based upon object size, storage capacity in an Autonomous System, and distance between the Autonomous Systems. Their formula is as follows:

Notation: the average number of hops that a request must traverse from all ASs is then

$$C_{(x)} = \sum_{i=1}^{I} \sum_{j=1}^{J} s_{ij} d_{ij}(x) \qquad (1)$$

where $d_{ij}(x)$ is the shortest distance to a copy of object j from AS_i under the placement x**,** J is the number of objects, I is the number of AS. S_{ij} is the number of bytes of storage in AS_j for the i^{th} object.

Given a target number of hops T, we ask if there is a placement x such that

$$\sum_{i=1}^{I} \sum_{j=1}^{J} s_{ij} d_{ij}(x) \leq T \qquad (2)$$

subject to

$$\sum_{i=1}^{I} \sum_{j=1}^{J} s_{ij} d_{ij}(x) \leq S_i \qquad i=1,2,3,...,I \qquad (3)$$

They have proved that this is a NP-hard problem. It means that for a large number of objects and ASs, it is not feasible to solve this problem optimally [34]. Based on this, they have adopted a similar approach to [32] which utilized a heuristic algorithm to solve the placement problem. The algorithms they evaluated were: Random, Popularity, Greedy-Single, Greedy-Global. Similarly, in [32], the authors

investigated: Purely local algorithms including MFUPlace, LRU Replacement, GreedyDual Replacement, and Cooperative placement algorithms including an optimal placement algorithm, simple near-optimal placement algorithm, Greedy placement algorithm, Amortized placement algorithm, and Hierarchical GreedyDual algorithm. Eventually, both [33] and [32] concluded that a cooperative approach is the best one. And [32] also identified that client access traffic pattern is a key challenge for replica placement. In [32], with their simulations are based on a Zipf [35] network model, they find Peer-to-Peer[9] is a good way out of this NP-hard problem and achieves optimal replica placement and replacement.

In [36], they consider this replica placement problem in different granularities. Similar to [32] that the authors established a cost model. They also believe that the cooperative placement and replacement strategy is a better approach. The major contribution of their work is to introduce a cluster-based replication strategy. Their comparison of states to maintain and computational cost amongst different mechanisms (per web site, per cluster and per URL) shows that a cluster-based replication schema is relatively good. In simulation of the web site MSNBC, the cluster-based replication outperformed the other strategies. In particular, their incremental clustering schema is very useful in improving content data availability during flash crowds for popular web site since it adapts well to user access patterns.

1.4.4.2.2 Discussions of replica placement algorithms criteria

A replica placement strategy decides what content is to be replicated and where, such that some objective function is optimized under a given traffic pattern and a set of resource constraints. In the objective function, the following metrics should be taken into consideration:

A. Reads: the rate of read accesses by a client to an object. This might also be reflected as the probability of an access to an object within time units.
B. Writes: the rate of write access by a client to an object.
C. Distance: the distance between a client and an original content/replica server, represented with a metric such as latency
D. Storage cost: the cost of storing an object at a replica sever. This might reflect the size of object, the throughput of the server, or the fact if a replica in the server
E. Content size: the size of object in bytes
F. Access time: A time stamp indicating the last time object was accessed at a replica server
G. Hit ratio: hit ratio of any replica along the path

In addition, the following constraint primitives can be added, such as: storage capacity of a replica server, load capacity of a replica server, node bandwidth capacity of a replica server, link capacity between a client and the replica server, number of replicas to be disseminated, original copy location, delay to be tolerant by a CDN, availability of certain object in a CDN. In [37], the authors have made an intensive study of many replica placement algorithms and proposed sophisticated metrics to evaluate different replica algorithms. Particularly, they have summarized all of their cost functions in their paper. This provides us comprehensive understanding of what constraints were considered in each schema of replica placement algorithm.

[9] Peer-to-Peer will be explained in the following chapter (chapter 2).

To sum up, replica placement and replacement has been well researched for CDN. Ultimately, a placement algorithm is about how to disseminate replicas under the constraints amongst resources and Quality of Service.

1.4.4.3 Replica management

How to maintain data consistency between the master copy of the content in the content server and replicas amongst replica servers is one of the most important questions for all CDNs. If there is a change occurs, updates of this object over the replica servers must be done. It is important that a content user not get a stale version of request content. From CDN's perspective, over head traffic of the updates should be minimized on the overlay network. This is called the replica or cache *coherency* or *consistency* problem, and is depicted in Figure 5. In the following subsections, I will explain two types of replica management strategies– strong consistency and weak consistency. There are many object attributes in HTTP [38], which can assist replica servers to maintain cache coherency.

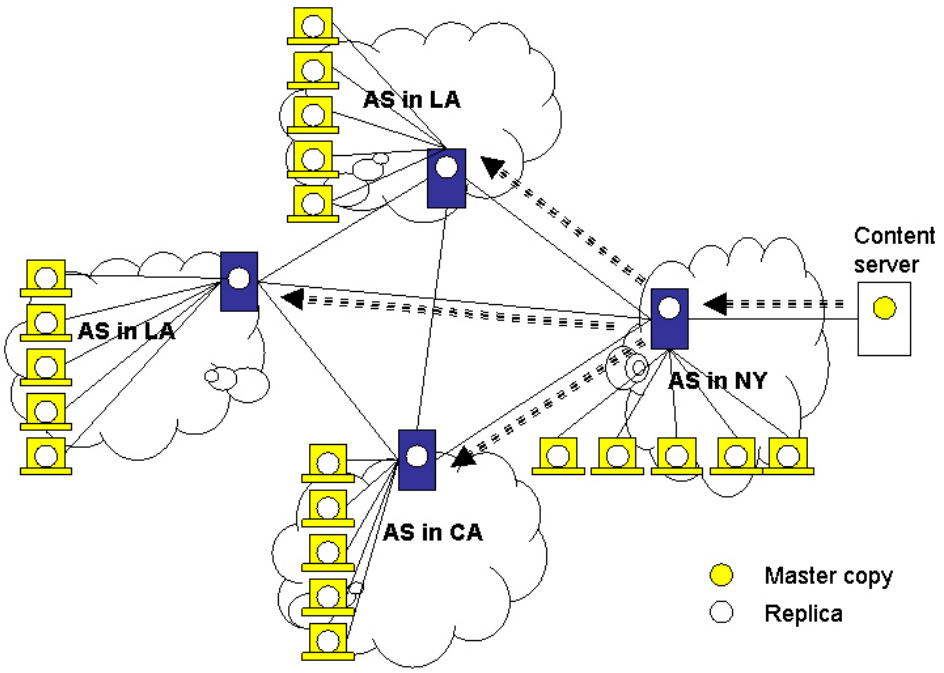


Figure 5. Replica consistency overview

1.4.4.3.1 Strong consistency

Client validation This approach is also called *polling every-time*. The client treats cached resources as potentially out-of-date on each access and sends an *If-Modified-Since* header with each access of the resources. This approach can lead to many 304 responses (the HTTP response code for "*Not Modified*") by server if the resource does not actually change.

Server invalidation Upon detecting a resource change, the server sends invalidation messages to all clients that have recently accessed and potentially cached the resource [49]. This approach requires a server to keep track of clients to use for invalidating cached copies of changed resources and can become cumbersome for a server when the number of clients is large, thus lead to a scalability problem. In addition, the lists themselves can become out-of-date causing the server to send invalidation messages to clients who are no longer caching the resource. Thus it causes unnecessary traffic.

Adaptive Leases The server employs a lease mechanism, which determines for how long it should propagate invalidates to the proxies [59]. This work also presents policies under which appropriate lease durations are computed so as to balance the trade-offs of state space overhead and control message overhead.

Propagation and Invalidation Combination Fei [60] proposed a smart propagation policy in a hybrid approach (propagation and invalidation).The rationale is to distinguish when to use unicast for invalidation and when to use multicast to propagate the updates. In this propagation policy, the author uses this notation: U is the object/document update rate at the origin content server and the total request rate is R. N is the number of replicas of this object. Є is the factor in the relative efficiency of unicast and multicast [61]. CDN chooses propagation for each object if the following inequality is true:

$$U * N^{1+\epsilon} < R \qquad (4)$$

Otherwise, use invalidation where $-0.34 < \epsilon < 0.30$. Intensive simulation results shows that this method significantly reduced the traffic generated during replica consistency.

1.4.4.3.2 Weak consistency

Adaptive TTL Similar to Time to Live for a packet in IPv4[51], the adaptive TTL [52] handles the problem by adjusting a document's time-to-live based on observations of its lifetime. Adaptive TTL takes advantage of the following facts; if a file has not been modified for a long time, it tends to stay unchanged. Thus, the time-to-live attribute to a document is assigned to be a percentage of the document's current "age", which is the current time minus the last modified time of the document. Studies [52] have shown that adaptive TTL can keep the probability of stale documents within reasonable ranges (<5%). Most proxy servers (i.e. [53], [54], [55]) use this mechanism.

However, there are several drawbacks with this expiration-based coherence [50]. First, users must wait for expiration checks to occur even though they are tolerant the staleness of the requested page. Second, if a user is not satisfied with the staleness of a returned document, they have no choice but to use a tool (Progma) to send a No-Cache request to load the entire document from its home site. Third, the mechanism provides no strong guarantee regarding document staleness. Forth, users can not specify the degree of staleness they are willing to tolerate. Finally, when the user aborts a document load, caches often abort a document load as well.

Piggyback Invalidation Authors of [56], [57], [58] proposed such a mechanism to improve the effectiveness of cache coherency. They have proposed three invalidation mechanisms as follows:

The Piggyback Cache Validation (PCV) [56] capitalizes on requests sent from the proxy cache to the server to improve coherency. In the simplest case, whenever a proxy cache has a reason to communicate with a server it piggybacks a list of cached, but potentially stale, resources from that server for validation.

The basic idea of the Piggyback Server Invalidation (PSI) mechanism [57]is for servers to piggyback on a reply to a proxy, the list of resources that have changed since the last access by this proxy. The proxy invalidates cached entries on the list and can extend the lifetime of entries not on the list.

They also proposed a hybrid approach which combines the PSI and the PCV techniques to achieve the best overall performance [58]. The choice of the mechanism depends on certain time parameter. If the time is small, then the PSI mechanism is used while the PCV mechanism is used to explicitly validate cache contents for longer interval.

The Distributed Object Consistency Protocol Researchers at HP proposed a protocol to enhance an HTTP cache control mechanism. This protocol focuses on two goals: to reduce response time and server demand. The Distributed Object Consistency Protocol [62] defines a new set of HTTP headers to provide object consistency between content origin servers and edge proxy servers. DOCP distributes the ability to serve objects authoritatively on behalf of a content provider throughout the network. This middle-ware like architecture actually represents the request client by its master and content server by its slave. The following picture depicts the overview of this architecture.

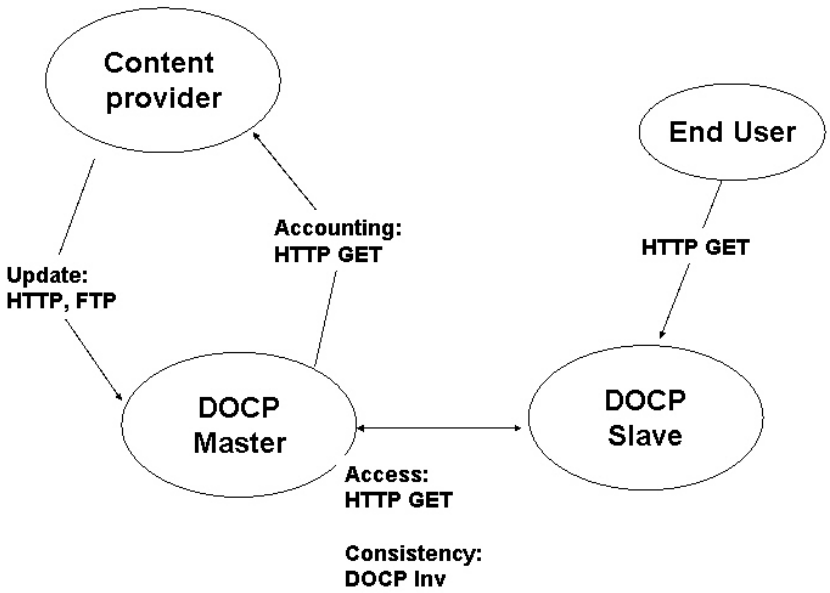


Figure 6. HP DOCP Architecture

1.4.4.4 Server location and request routing

In the location and routing system of a CDN, the location and routing system must be able to serve the client request and redirect the request to a replica server which is located as near as possible to the requesting client. In fact, server location and request routing are two aspect of the same problem of implementing a request service. From a client's perspective, we can formulate it as a server location problem; from CDN's perspective, we shall form the request routing problem for a client who requests certain content. In the following sections, I will explain these two views one after another.

1.4.4.4.1 Server location

Similar to the problem of placement of replica servers and content data replicas, how a client can allocate the best server in terms of proximity metrics and replica server load is another important issue in a CDN system. Since a replica of specific content is stored in some server in most CDNs, this implies that the client will locate the replica if it selects the right server.

1.4.4.4.1.1 Multicast vs. Agent

These two techniques can be considered as reactive and proactive approaches. In the former solution, when a client needs to find one server the CDN can send this request to all its replica servers in certain Multicast group. Server type will be cataloged by service. In the case, the client chooses the server who generates the quickest response from amongst that group. The disadvantage is high overhead on these messages sent to all the servers in a group or multi groups. This has been studied in [43]. Unlike the former approach, the Agent approach is more efficient. We can use an agent to probe different servers periodically and to maintain a list of servers which knows the most up-to-date load information for each server. And the agent can communicate with their own protocols to co-ordinate with each other in different locations. When a client requests a type of servers, the agent will select a right server for the client.

1.4.4.4.1.2 Routing layer vs. application layer

In [44], the authors proposed a way of using Anycasting to select the nearest server for a client. However, it assumes all servers have the same services. Thereby selection of different services can not be done unless policy constraints programming in the routers. On the contrary, application layer location services can provide better service differentiation, load information, and even bandwidth information for the clients. Similar to previous approach, an agent can act as monitor for the routing-layer traffic and decide to send updates to the database at rendezvous point. Updates will be on demand by the agent, thus many traffic overheads can be reduced [45]. However, the potentially can lead to a single point of failure when larger number of client are sending requests to the CDN.

Most important metrics in server selection are: distance between the client and replica server, server load, service type, and bandwidth on the link. The above techniques are quite widely used in today's' CDNs.

1.4.4.4.2 Request routing

In request routing, we address the problem of deciding which replica server can best service a given client request, in terms of the metrics. These metrics can be, for example, replica server load (where we choose the replica server with the lowest load), end-to-end latency (where we choose the replica server that offers the shortest response time to the client), or distance (where we choose the replica server that is closest to the client). In according to IETF's classifications [46], there are four catalog and eighteen types of request routing mechanisms. Figure 7 depicts all of them. Since request routing has been well studied and standardized, I will just summarize these results in the following paragraphs. For the detailed reference please see RFC 3568 [163].

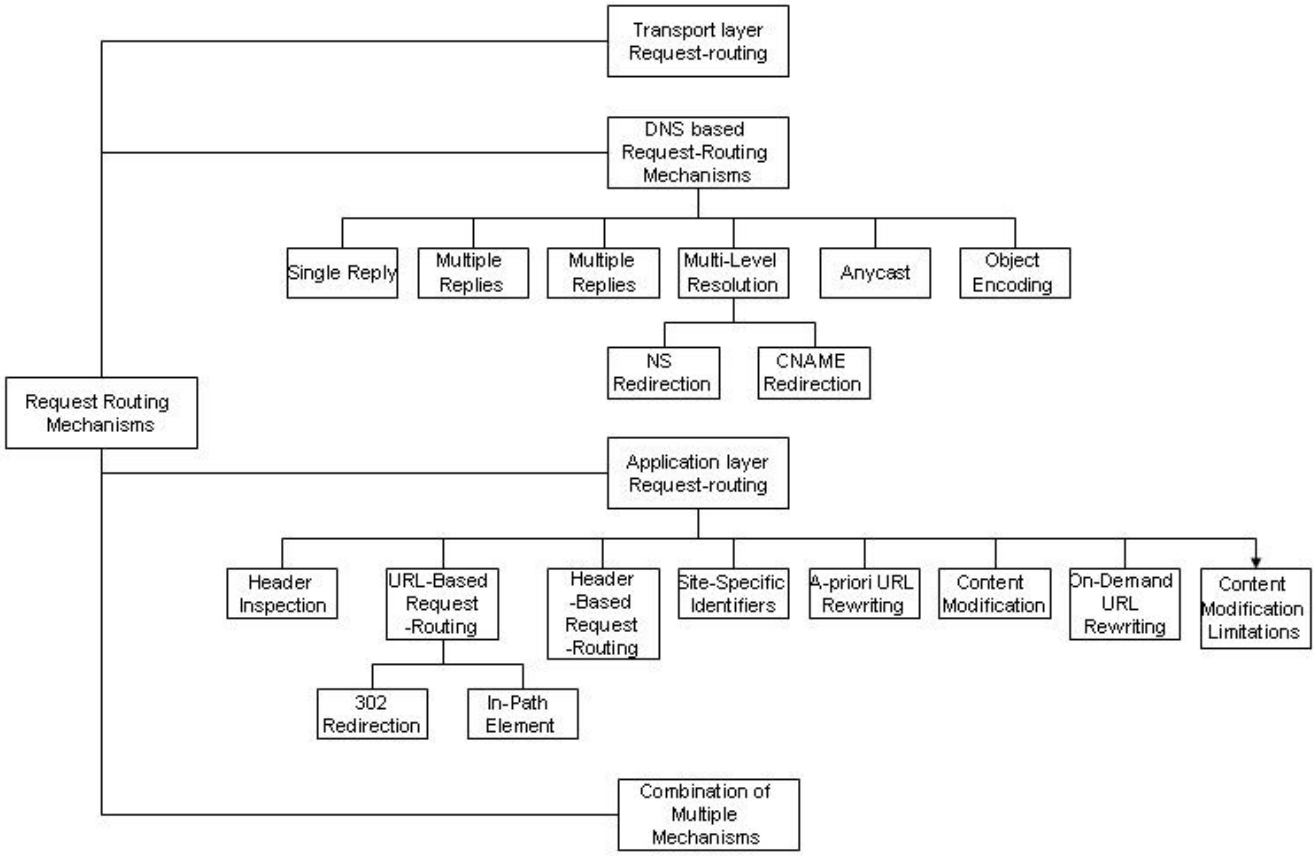


Figure 7. Content request routing mechanisms

1.4.4.4.2.1 Transport-Layer Request-Routing

At the transport-layer finer levels of granularity can be achieved by close inspection of the client's requests. In this approach, the Request-Routing system inspects the information available in the first packet of the client's request to make surrogate selection decisions. The inspection of the client's requests provides data about the client's IP address, port information, and layer 4 protocols. The acquired data could be used in combination with user-defined policies and other metrics to determine the election of a surrogate that is best suited to serve the request.

In general, the forward-flow traffic (client to newly selected surrogate) will flow through the surrogate originally chosen by DNS. The reverse-flow (surrogate to client) traffic, which normally transfers much more data than the forward flow, would typically take the direct path.

The overhead associated with transport-layer Request-Routing is better suited for long-lived sessions such as FTP [161] and RTSP [162]. However, it also could be used to direct clients away from overloaded surrogates.

In general, transport-layer Request-Routing can be combined with DNS based techniques. As stated earlier, DNS based methods resolve clients requests based on domains or sub domains based on the client's DNS server's IP address. Hence, the

DNS based methods could be used as a first step in deciding on an appropriate surrogate with more accurate refinement made by the transport-layer Request-Routing system.

1.4.4.4.2.2 Single Reply

In this approach, the DNS server is authoritative for the entire DNS domain or a sub domain. The DNS server returns the IP address of the best surrogate in an A record to the requesting DNS server. The IP address of the surrogate could also be a virtual IP (VIP) address of the best set of surrogates for requesting DNS server.

1.4.4.4.2.3 Multiple Replies

In this approach, the Request-Routing DNS server returns multiple replies such as several records for various surrogates. Common implementations of client site DNS server's cycle through the multiple replies in a Round-Robin fashion. The order in which the records are returned can be used to direct multiple clients using a single client site DNS server.

1.4.4.4.2.4 Multi-Level Resolution

In this approach multiple Request-Routing DNS servers can be involved in a single DNS resolution. The rationale of utilizing multiple Request-Routing DNS servers in a single DNS resolution is to allow one to distribute more complex decisions from a single server to multiple, more specialized, Request-Routing DNS servers. The most common mechanisms used to insert multiple Request-Routing DNS servers in a single DNS resolution is the use of NS and CNAME records. An example would be the case where a higher level DNS server operates within a region, directing the DNS lookup to a more specific DNS server within that region to provide a more accurate resolution.

1.4.4.4.2.5 NS Redirection

A DNS server can use NS records to redirect the authority of the next level domain to another Request-Routing DNS server. The, technique allows multiple DNS server to be involved in the name resolution process. For example, a client site DNS server resolving a.b.example.com would eventually request a resolution of a.b.example.com from the name server authoritative for example.com. The name server authoritative for this domain might be a Request-Routing NS server. In this case the Request-Routing DNS server can either return a set of “A” records or can redirect the resolution of the request a.b.example.com to the DNS server that is authoritative for example.com using NS records.

One drawback of using NS records is that the number of Request-Routing DNS servers is limited by the number of parts in the DNS name. This problem results from the DNS policy that causes a client site DNS server to abandon a request if no additional parts of the DNS name are resolved in an exchange with an authoritative DNS server.

A second drawback is that the last DNS server can determine the TTL of the entire resolution process. Basically, the last DNS server can return in the authoritative section of its response its own NS record. The client will use this cached NS record for further request resolutions until it expires.

Another drawback is that some implementations of *bind* voluntarily cause timeouts to simplify their implementation in cases in which a NS level redirect points to a name server for which no valid A record is returned or cached. This is especially a problem if the domain of the name server does not match the domain currently resolved, since in this case the A records, which might be passed in the DNS response, are discarded for security reasons. Another drawback is the added delay in resolving the request due to the use of multiple DNS servers.

1.4.4.4.2.6 CNAME[10] Redirection

In this scenario, the Request-Routing DNS server returns a CNAME record to direct resolution to an entirely new domain. In principle, the new domain might employ a new set of Request-Routing DNS servers. One disadvantage of this approach is the additional overhead of resolving the new domain name. The main advantage of this approach is that the number of Request-Routing DNS servers is independent of the format of the domain name.

1.4.4.4.2.7 Anycast

Anycast [44] is an network service that is applicable to networking situations where a host, application, or user wishes to locate a host which supports a particular service but, if several servers utilize the service, it does not particularly care which server is used. In an Anycast service, a host transmits a datagram to an Anycast address and the network is responsible for providing best effort delivery of the datagram to at least one, and preferably only one, of the servers that accept Datagrams for the Anycast address.

The motivation for Anycast is that it considerably simplifies the task of finding an appropriate server. For example, users, instead of consulting a list of servers and choosing the closest one, could simply type the name of the server and be connected to the nearest one. By using Anycast, DNS resolvers would no longer have to be configured with the IP addresses of their servers, but rather could send a query to a well-known DNS Anycast address. Furthermore, to combine measurement and redirection, the Request-Routing DNS server can advertise an Anycast address as its IP address. The same address is used by multiple physical DNS servers. In this scenario, the Request-Routing DNS server that is the closest to the client site DNS server in terms of OSPF and BGP routing will receive the packet containing the DNS resolution request. The server can use this information to make a Request-Routing decision. Drawbacks of this approach are:

- The DNS server may not be the closest server in terms of routing to the client.
- Typically, routing protocols are not load sensitive. Hence, the closest server may not be the one with the least network latency.
- The server load is not considered during the Request-Routing process.

1.4.4.4.2.8 Object Encoding

[10] CNAME stands for canonical name. (CNAME) A host's official name as opposed to an alias. The official name is the first hostname listed for its Internet address in the hostname database, /etc/hosts or the Network Information Service (NIS) map hosts.byaddr ("hosts" for short). A host with multiple network interfaces may have more than one Internet address, each with its own canonical name (and zero or more aliases). You can find a host's canonical name using nslookup if you say set querytype=CNAME and then type a hostname.

Since only DNS names are visible during the DNS Request-Routing, some solutions encode the object type, object hash, or similar information into the DNS name. This might vary from a simple division of objects based on object type (such as images.a.b.example.com and streaming.a.b.example.com) to a sophisticated schema in which the domain name contains a unique identifier (such as a hash) of the object. The obvious advantage is that object information is available at resolution time. The disadvantage is that the client site DNS server has to perform multiple resolutions to retrieve a single Web page, which might increase rather than decrease the overall latency.

1.4.4.4.2.9 DNS Request-Routing Limitations

This section lists some of the limitations of DNS based Request-Routing techniques.

- DNS only allows resolution at the domain level. However, an ideal request resolution system should service requests on a per object level.
- In DNS based Request-Routing systems servers may be required to return DNS entries with short time-to-live (TTL) values. This may be needed in order to be able to react quickly in the face of outages. This in turn may increase the volume of requests to DNS servers.
- Some DNS implementations do not always adhere to DNS standards. For example, many DNS implementations do not honor the DNS TTL field.
- DNS Request-Routing is based only on knowledge of the client DNS server, as client addresses are not relayed within DNS requests. This limits the ability of the Request-Routing system to determine a client's proximity to the surrogate.
- DNS servers can request and allow recursive resolution of DNS names. For recursive resolution of requests, the Request-Routing DNS server will not be exposed to the IP address of the client's DNS server. In this case, the Request-Routing DNS server will only be exposed to the address of the DNS server that is recursively requesting the information on behalf of the client's site DNS server. For example, imgs.example.com might be resolved by a CN, but the request for the resolution might come from dns1.example.com as a result of the recursion.
- Users that share a single client site DNS server will be redirected to the same set of IP addresses during the TTL interval. This might lead to overloading of the surrogate during a flash crowd unless different sites got different answers.
- Some implementations of bind can cause DNS timeouts to occur while handling exceptional situations. For example, timeouts can occur for NS redirections to unknown domains.

DNS based request routing techniques can suffer from serious limitations. For example, the use of such techniques can overburden third party DNS servers, which should not be allowed. In RFC 2782 [164], provides warnings on the use of DNS for load balancing. Readers are encouraged to read the RFC for better understanding of these limitations.

1.4.4.4.2.10 Application-Layer Request-Routing

Application-layer Request-Routing systems perform deeper examination of client's packets beyond the transport layer header. Deeper examination of client's packets provides fine-grained Request-Routing control down to the level of individual objects.

The process could be performed in real time at the time of the object request. The exposure to the client's IP address combined with the fine-grained knowledge of the requested objects enable application-layer Request-Routing systems to provide better control over the selection of the best surrogate.

1.4.4.4.2.11 Header Inspection

Some application level protocols such as HTTP, RTSP, and SSL [165] provide hints in the initial portion of the session about how the client request must be directed. These hints may come from the URL of the content or other parts of the MIME request header such as Cookies.

1.4.4.4.2.12 URL-Based Request-Routing

Application level protocols such as HTTP and RTSP describe the requested content by its URL. In many cases, this information is sufficient to disambiguate the content and suitably direct the request. In most cases, it may be sufficient to make Request-Routing decision just by examining the prefix or suffix of the URL.

1.4.4.4.2.13 302 Redirection

In this approach, the client's request is first resolved to a virtual surrogate. Sequentially, the surrogate returns an application-specific code such as the 302 (in the case of HTTP or RTSP) to redirect the client to the actual delivery node.

This technique is relatively simple to implement. However, the main drawback of this method is the additional latency involved in sending the redirect message back to the client and the client having to resolve a new address.

1.4.4.4.2.14 In-Path Element

In this technique, an In-Path element is present in the network in the forwarding path of the client's request. The In-Path element provides transparent interception of the transport connection. The In-Path element examines the client's content requests and performs Request-Routing decisions.

The In-Path element then splices the client connection to a connection with the appropriate delivery node and passes along the content request. In general, the return path would go through the In-Path element. However, it is possible to arrange for a direct return by passing the address translation information to the surrogate or delivery node through some proprietary means.

The primary disadvantage with this method is the performance implications of URL-parsing in the path of the network traffic. However, it is generally the case that the return traffic is much greater than the forward traffic.

The technique allows for the possibility of partitioning the traffic among a set of delivery nodes by content objects identified by URLs. This allows object-specific control of server loading. For example, requests for non-cacheable object types may be directed away from a cache.

1.4.4.4.2.15 Header-Based Request-Routing

This technique involves using HTTP attributes such as Cookie, Language, and User-Agent, in order to select a surrogate.

Cookies can be used to identify a customer or session by a web site. Cookie based Request-Routing provides content service differentiation based on the client. This approach works provided that the cookies belong to the client. In addition, it is possible to direct a connection from a multi-session transaction to the same server to achieve session-level persistence.

The language header can be used to direct traffic to a language-specific delivery node. The user-agent header helps identify the type of client device. For example, a voice-browser, PDA, or cell phone can indicate the type of delivery node that has content specialized to handle the content request.

1.4.4.4.2.16 Site-Specific Identifiers

Site-specific identifiers help authenticate and identify a session from a specific user. This information may be used to direct a content request.

An example of a site-specific identifier is the SSL Session Identifier. This identifier is generated by a web server and used by the web client in succeeding sessions to identify itself and avoid an entire new security authentication exchange. In order to inspect the session identifier, an In-Path element would observe the responses of the web server and determine the session identifier which is then used to associate the session to a specific server. The remaining sessions are directed based on the stored session identifier.

1.4.4.4.2.17 Content Modification

This technique enables a content provider to take direct control over Request-Routing decisions without the need for specific switching devices or directory services in the path between the client and the origin server. Basically, a content provider can directly communicate to the client the best surrogate that can serve the request. Decisions about the best surrogate can be made on a per-object basis or it can depend on a set of metrics. The overall goal is to improve scalability and the performance for delivering the modified content, including all embedded objects.

In general, the method takes advantage of content objects that consist of a basic structure that includes references to additional, embedded objects. For example, most web pages consist of an HTML document that contains plain text together with some embedded objects, such as GIF or JPEG images. The embedded objects are referenced using embedded HTML directives. The embedded HTML directives direct the client to retrieve the embedded objects from the origin server. A content provider could modify references to embedded objects such that they could be fetched from the best surrogate. This technique is also known as URL rewriting.

Content modification techniques must not violate the architectural concepts of the Internet [48]. Special considerations must be made to ensure that the task of modifying the content is performed in a manner that is consistent with RFC 3238 [48]; it specifies the architectural considerations for intermediaries that perform operations or modifications on content.

The basic types of URL rewriting are discussed in the following subsections.

1.4.4.4.2.17.1 A-priori URL Rewriting

In this scheme, a content provider rewrites the embedded URLs before the content is placed on the origin server. In this case, URL rewriting can be done either manually or by using software tools that parse the content and replace embedded URLs.

A-priori URL rewriting alone does not allow consideration of client specifics for Request-Routing. However, it can be used in combination with DNS Request-Routing to direct related DNS queries into the domain name space of the service provider. Dynamic Request-Routing based on client specifics are then done using the DNS approach.

1.4.4.4.2.17.2 On-Demand URL Rewriting

On-Demand or dynamic URL rewriting, modifies the content when the client request reaches the origin server. At this time, the identity of the client is known and can be considered when rewriting the embedded URLs. In particular, an automated process can determine, on-demand, which surrogate would serve the requesting client best. The embedded URLs can then be rewritten to direct the client to retrieve the objects from the best surrogate rather than from the origin server.

1.4.4.4.2.17.3 Content Modification Limitations

Content modification as a Request-Routing mechanism suffers from many limitation. For example:

- The first request from a client to a specific site must be served from the origin server.
- Content that has been modified to include references to nearby surrogates rather than to the origin server should be marked as non-cacheable. Alternatively, such pages can be marked to be cacheable only for a relatively short period of time. Rewritten URLs on cached pages can cause problems, because they can become outdated and point to surrogates that are no longer available or no longer good choices.

1.4.4.4.2.18 Combination of Multiple Mechanisms

There are environments in which a combination of different mechanisms can be beneficial and advantageous over using one of the proposed mechanisms alone. The following example illustrates how the mechanisms can be used in combination. A basic problem of DNS Request-Routing is the resolution granularity that allows resolution on a per-domain level only. A per-object redirection cannot easily be achieved. However, content modification can be used together with DNS Request-Routing to overcome this problem. With content modification, references to different objects on the same origin server can be rewritten to point into different domain name spaces. Using DNS Request-Routing, requests for those objects can now dynamically be directed to different surrogates.

1.4.4.5 Self-organization

User, content provider, and their Internet Service providers are the three major actors in every CDN system. User wants fast access of the content, content provider wants high availability of their content, and the ISP wants no network congestion or minimal bandwidth consumption for the traffic over their network. Therefore a good CDN

system must find the best trade-offs to meet all above requirements. Example of such systems include Cisco's Boomerang algorithm in their patented Self-organizing Distributed Architecture (SODA). However, this is what they say:

Cisco Content routers utilize an extremely fast site-selection algorithm (Boomerang) to have each site respond to a Domain Name System query with the IP address of a server at the site. The Boomerang process selects the site that has the least amount of network delay between the site and the client's DNS server at that exact instant in time. The first response through the network wins the race and is used by the client to connect to a server at the requested web site. Cisco's content networking solutions for enterprise CDNs also use SODA technology. SODA enables the system to intelligently route requests and deliver content to the end user. The SODA algorithm learns about the network, thus performance improves with time and usage. When a browser selects content from a specific Web site, the content delivery network will look at the source IP address and redirect the request to the optimal content engine.

Boomerang and SODA attempt to ensure the fastest delivery of content regardless of location, providing the highest availability and site response. SODA provides for the ability to redirect user requests to the best content router by allowing CDN devices to automatically organize themselves into a single cooperating system. Because the CDN devices collectively determine the fastest route, they also provide the fastest delivery of high-bandwidth content.[76]

In the Location and routing system of Cisco's CDN, The functions as I described in section 1.2.4.4.2. In Boomerang process, the most interesting part is that the SODA algorithm learns about the underlying network design and performance to the end user with usage improves. Thus when new requests arrive for certain content, the content router selects a replica server for the client in terms of metrics such as presence of content, geographical location, originating network, current network conditions, and current replica server health. Their content router actually integrates a redirection server and layer three router in this one cabinet. Thus, it does congestion measurement for the underlying network before determining the redirection of each client request content replica server. Thus it optimizes the layer 3 performance by avoiding congestion on the link and this certainly benefits the client access performance. However, in their architecture, the self-organization replies on their proprietary infrastructure i.e. Cisco CR-4450 content router [77].

1.5 Discussion

After examining the problems and approaches of Server placement, Replica placement, Replica management, Server location and request routing in CDNs, we find that there are some issues are very interesting for future CDNs.

1.5.1 Large content for large numbers of users

Nowadays, large content (i.e. movie, music and video-on-demand etc.) publishing becomes more and more popular on web sites just because of high demand from Internet users. Therefore content distribution networks face a challenge in delivering such large content objects. On one hand, this requires lots of bandwidth, and predictable transmission delay to achieve high data availability. On the other hand, the number of content providers turning to CDNs to better service their customers is

growing rapidly. Thereby, large and highly dynamic content distribution will raise many issues concerning the design and architecture of CDNs in the future, such as:

- How to replicate the large amount of content?
- How to use limited bandwidth to transmit this content?
- How to scale up to large numbers of users who want to access the large size of content?
- How to distribute the load of CDN when frequent accessed the content sites appear?...

In the case of the Spirit spaceship landing on Mars, NASA published many pictures and videos on their web site. The high hit rate [63], [64] indicates the challenges CDN will have when large number of users request large content objects. Content Distribution network Interconnection [67] [74] has been studied for some time. Their solution is to connect CDNs to interconnect CDNs or peering CDNs in terms of many concerns such as CDN performance, security, accounting [67], and economic reasons [68]. I dare to guess that the delivery in the case of the Spirit spaceship landing Mars, the eTouch [69], Speedra [70] and Sprint[71] had to address this scenario to distributed NASA's traffic for this Mars event. But this solution is an very expensive one as I mentioned in section 5.1.3. Is there any better way to handle this large volume of traffic? I believe this should be a research focus in the next generation CDNs.

1.5.2 Denial of service attack

When we are talking about normal access request for the content server, there is another fact which should not be ignored, that is a type of abnormal access request, i.e., as part of a Denial of Service attack (DoS). These attacks typically flood a network or server with bogus request packets, rendering it unavailable to handle legitimate requests. Despite increased awareness about security issues, denial of service attacks remain a challenging problem. According to a Computer Security Institute survey, for example, the number of respondents indicating their sites had been the victim of a DoS attack rose from 27% in 2000 to 38% in 2001 [72] News about DoS can be found in the section of "*Selected news reports/interviews/panel discussions*" of [73]. The key problem is that Distributed Denial of Service (DDoS) utilized legitimate access method (such as the attacker sends TCP SYN packets which can be accepted by CDN servers) to overwhelm the content server or replica servers in CDN. Study [72] and CDI [66] choose the server side techniques to solve this problem. As we know, all security mechanisms incur overhead to do checking and action. Can we have some more cost-effective solution of this problem?

1.5.3 Scalability issue

The magnitude of the Internet population is still increasing exponentially. How well can a CDN or CDNs work when the number of access clients continuously increases is the problem for today's CDNs. CDI [66] connects many CDNs to increase the scalability by peering affords. The rationale of doing so is to expand the number of the replica servers and offer better load balancing amongst them. Management overhead becomes inevitable when coordination happens amongst them. Can we have a simpler solution than that?

1.5.4 Self-organization in next generation of CDNs

For large content distribution, large bandwidth consumption is inevitable. Thereby making the best trade-off amongst decreasing access latency, maximizing content data

availability, and optimizing network performance becomes more difficult than small and middle size content delivering in today's CDNs. Together with new techniques such as Peer-to-Peer, and swarm intelligence, what are the requirements should be implemented in self-organization of CDNs? This should be answered in the next generation CDN systems.

Chapter 2 Introduction to Peer-to-Peer

2.1 A definition of P2P

There are many definitions for Peer-to-Peer technology. In this paper, use this definition from The Free Encyclopedia [166]

*Generally, a Peer-to-*Peer (or *P2P*) *computer network refers to any network that does not have fixed clients and servers, but a number of* peer *nodes that function as both clients and servers to the other nodes on the network. This model of network arrangement is contrasted with the client-server model. Any node is able to initiate or complete any supported transaction. Peer nodes may differ in local configuration, processing speed, network bandwidth, and storage quantity. Popular examples of P2P are file sharing-networks.*

According to this definition, we can understand that equivalence is the most important property of such systems. Another important property is decentralization of such systems. The equivalence is reflected by the decentralization in a P2P system or application because P2P refers to a class of systems or applications that employ distributed resources to perform a crucial function in a decentralized manner[79]. The reason why P2P computing has becomed so famous is because that it solves the problems of utilizing many distributed systems in a better way compared with client-server computing. Moreover, it brings significant conveniency to its users's for their many different communication demands at work and at play for example, online collaboration, music-sharing, and online gaming, instant messaging etc. This makes Peer-to-Peer one of the hottest Internet trends!

2.2 Problem domains and well-known approaches

Differing from CDN, P2P technology is type of communication/computation pattern [80]. It means that P2P computational mechanisms could be applied in many subareas in the realm of distributed systems such as distributed computing of financial and biotechnology, file-sharing, collaboration of fault tolerant and real time constraints, communication middleware. In this paper, we will not describe all the problems which P2P can address in these subareas and disiplines. Instead, we will focus its properties. Following this, we select the features of P2P which can be used in our PlentyCast design. In the following subsections, I use the P2P characteristics described by [79] as a base, then extend their discussion.

2.2.1 Decentralization

P2P models question the wisdom of storing and processing data only on centralized servers and accessing the content via request-response protocols. In traditional client-server models, the information is concentrated in centrally located servers and distributed through networks to client computers that act primarily as user interface devices. Such centralized systems are ideal for some applications and tasks. For example, access rights and security are traditionally more easily managed in centralized systems. However, the topology of the centralized systems may result in inefficiencies, bottlenecks, and wasted resources. Furthermore, although hardware performance and cost have improved, centralized repositories are expensive to set up

and hard to maintain. They require human intelligence to build, and to keep the information they contain relevant and current.

One of the powers of decentralized systems is the emphasis on the user's ownership and control of the data and resources. In my viewpoint, managing resources and data in centralized systems have more overheads, take slower response, and have greater probability of error occurring comparing with the user's ownership of the resources and data if there is a long distance existing between the user and the content objects. However, in full decentralized systems, every peer is an equal participant. This makes design difficult because there is no centralized server with a global view of all the peers in the network or the files they provide. This is why many P2P systems are built in hybrid approach as in Napster[92], where there is a central directory of the files but peers will download the file from other nodes directly.

In full decentralized file systems, Freenet[42] and Gnutella[40], a big problem is discovery of the nearest elements of network. For instance, Gnutella must learn one IP address of a peer then the new node can send Ping packets to discovery the rest of the peers and cache the host list of the peers it can reach. Therefore, the problem is to make a trade-off between pure P2P and hybrid P2P in terms of the focus of systems features such as file sharing, collaboration and computation, or platform. This categorization has a direct effect on the self-organization and scalability[11] of a system because of the loose coupling to any infrastructure.

2.2.2 Scalability

An immediate benefit of decentralization is improved scalability[81]. Scalability is limited by the amount of centralized operations (e.g. synchronization and coordination) that needs to be performed, the amount of state that needs to be maintained, the inherent parallelism in an application, and programming model which is used to represent the computation.

There are many attempts to attack the scalability problem. Napster attacked the scalability problem by having the peers directly download music files from the peers that possess the requested document. As a result, Napster was able to scale to over 6 million users at the peak of its service. In contrast, the SETI@home [90] focuses on a task that is almost completely parallel. It harnesses the computer power that is available over the Internet to analyze data collected from its telescopes with the goal of searching for extraterrestrial life forms. The SETI@home has close to 3.5 million users so far.

Content Addressable Network (CAN) [82], Chord [83], Oceanstore [84], and PAST [85] dictate a consistent mapping between the object key and hosting node. Therefore, an object is always reachable as long as the host is connected. In such an overlay network, each node only maintains the address of a small number of other nodes. This limits the state information to be maintained for each node, thus scalability is increased. It is said that they can scale to billions of users, millions of servers, and 10^{13} records. The trade-off between scalability and self-organization shall be investigated in this case.

[11] Self-organization will be discussed in 2.2.3, and scalability will be discussed in 2.2.2

2.2.3 Self-organization

Self-organization is defined as "a process where the organization (constraints and redundancy) of a system spontaneously increases, i.e., without this increase being controlled by the environment or an encompassing or otherwise external system"[86] In P2P systems, self-organization is needed because of scalability, fault resilient[12], intermittent connection of resources, and the cost of ownership[13]. Since unpredictable numbers of users access the network in an ad hoc manner, adaptation should be implemented to handle the changes caused by peers connecting and disconnecting from system. It can also be a very important feature when a peer discovers the network. There are additional concerns in self-organization is about how to adapt the variance of network latency and bandwidth other than intermittent peer availability. In this case self-organization shall be embedded in the lower level communication APIs of the platform in a P2P system.

There are a number of academic systems and products that address self-organization, such as OceanStore, Pastry, Tapestry, Chord, and CAN.

In OceanStore [84], self-organization is applied to location and routing infrastructure. Because of intermittent peer availability, as well as variances in network latency and bandwidth, the infrastructure is continuously adapting its routing and location support.

In Pastry[88], self-organization is handled through protocols for node arrivals and departures based on a fault-tolerant overlay network. Client requests are guaranteed to be routed in less steps on average than the worst case. Also, file replicas are distributed and storage is randomized for load balancing.

In [87] , the authors proposed a secondary overlay to be layered on top of CAN[82], Chord[83], Tapestry [89], Pastry [88] that exploits knowledge of the underlying network characteristics. The secondary overlay builds a location layer between "supernodes," nodes that are situated near network access points, such as gateways to administrative domains. By associating local nodes with their nearby "supernode," messages across the wide-area can take advantage of the highly connected network infrastructure between these supernodes to shortcut across distant network domains, greatly reducing point-to-point routing distance and reducing network bandwidth usage.

2.2.4 Anonymity

An important goal of anonymity is to allow people to use the systems without concern for legal or other ramifications. Ultimately, it will make censorship of digital content impossible. At this point, content creator, participant, overlay operators, and content itself are not identical. Furthermore there are 6 technical approaches so far. They are multicast, spoofing the sender's address, identity spoofing, covert path, intractable aliases, and non-voluntary placement. There are more actors to be identified with respect to anonymity. Publisher, reader, server, and document are the main actors who can be anonymous [42]. In P2P, how anonymous the operator, the participant, and content creators should be one of the major problem to be resolved because one of the

[12] Fault resilient will be explain in section 2.2.9.
[13] Cost of ownership will be explained in section 2.2.5

business requirements to enable the content provide to know who has accessed his or her content.

2.2.5 Cost of ownership

One of premises of P2P computing is shared ownership. It reduces the cost of owning the system, the content, and maintenance. It means that the management overheads of controlling resources and data are reduced comparing with the client-server paradigm. From Napster [92], SETI@home [90] till BitTorrent [91], this is always to the advantage of P2P file sharing applications. In P2P, we will take use of this characteristics to enable a large number of computers join the P2P overlay network. The direct consequence of content sharing is that the operator can concentrate more on its services instead of infrastructure maintenance and expansion. However, how the peer and the server cost shall be shared is an important issue, because fairness is the basic principle that should be apply. The degree of cost sharing reflects the decentralization and self-organization.

2.2.6 Ad hoc connectivity

Their ad hoc nature strongly affects all classes of P2P systems. Basically, this means that there are some nodes in the network that may be available all the time, but some are only part of the time, and some of them are not available at all.

According to Mojo Nation P2P network measurements, 80%-84% of nodes fell into the availability group of disconnected "one time, less than one hour", and 16%-20% fell into the remaining longer than one hour, and significant fraction stayed connected less than 24 hours before being permanently disconnected [93]. It is considered an exception in traditional networks for a node to be not available, but in P2P system this will be very common. Understanding this is very important. Therefore, in content sharing P2P systems and applications, users expect to be able to access content intermittently, subject to the connectivity of the content providers. In systems with higher guarantees, such as service-level agreements, the ad-hoc nature is reduced by redundant service providers, but parts of the providers servers(and hence content) may be unavailable. How to handle intermittent availability of certain participants is a major problem to be resolved. Replicating the content and or reallocating the content will be challenging in such a dynamic environment. This directly effect the system design for fault resiliency[14] and self-organization.

Furthermore, not everything will be connected to the Internet. Even under these circumstance, ad hoc groups of people form ad hoc networks in order to collaborate. The supporting ad-hoc networking infrastructures, such as 802.11b, Bluetooth, and infrared, have only a limited radius of accessibility. Therefore, both P2P systems and applications need to be designed to tolerate sudden disconnection and ad hoc additions to groups of peers.

2.2.7 Performance

Performance is a significant concern in P2P systems. P2P systems aim to improve performance by aggregating distributed storage capacity (e.g., Napster, Gnutella) and computing cycles (e.g., SETI@home) of devices spread across a network. Because of the decentralized nature of these models, performance is influenced by three types

[14] Fault resilient will be explained in section 2.2.9

of resources: processing, storage, and networking. In particular, networking delays can be significant in wide area networks. Bandwidth is a major factor when a large number of messages are propagated in the network and large amounts of files are being transferred among many peers. This limits the scalability of the system. Performance in this context does not emphasize millisecond performance, but rather tries to answer questions of how long it takes to retrieve a file or how much bandwidth a query will consume. In centrally coordinated systems (e.g., Napster, SETI@home, and BitTorrent) coordination between peers is controlled and mediated by a central server, although the peers may also later contact each other directly. This makes these systems vulnerable to the problems facing centralized servers. To overcome the limitations of a centralized coordinator, different hybrid P2P architectures [94] have been proposed to distribute the functionality of the coordinator in multiple indexing servers that cooperate with each other to satisfy user requests. DNS is another example of a hierarchical P2P system that improves performance by defining a tree of coordinators, with each coordinator responsible for a peer group. In BitTorrent, trackers are designed to coordinate the peers' downloading and upload metrics. Communication between peers in different groups is achieved through a higher level coordinator. In decentralized coordinated systems such as Gnutella and Freenet, there is no central coordinator; communication is handled individually by each peer. Typically, they use message forwarding mechanisms that search for information and data. The problem with such systems is that they end up sending a large number of messages over many hops from one peer to another. Each hop contributes to an increase in the bandwidth on the communication links and to the time required to get results for the queries. The bandwidth for a search query is proportional to the number of messages sent, which in turn is proportional to the number of peers that must process the request before finding the data[98]. There are three key approaches to optimize performance: replication, caching, and intelligent routing.

2.2.7.1 Replication

Replication puts copies of objects/files closer to the requesting peers, thus minimizing the distance between the peers requesting and providing the objects. Changes to data objects have to be propagated to all the object replicas. Oceanstore uses an update propagation scheme based on conflict resolution that supports a wide range of consistency semantics. The geographic distribution of the peers helps to reduce congestion of both the peers and the network. In combination with intelligent routing, replication helps to minimize the delay by sending requests to closely located peers. Replication also helps to cope with the disappearance of peers. Because peers tend to be personal computer (PCs) rather than dedicated servers, there is no guarantee that the peers won't be disconnected from the network arbitrary. The key is replication of redundant data across the dynamic overlay in order to achieve maximal object data availability.

2.2.7.2 Caching

Caching reduces the path length required to fetch a file/object and therefore the number of messages exchanged between the peers. Reducing such transmissions is important because the communication latency between the peers is a serious performance bottleneck facing P2P systems. In Freenet for example, when a file is found and propagated to the requesting node, the file is cached locally in all the nodes in the return path. More efficient caching strategies can be used to cache large

amounts of data infrequently. The goal of caching is to minimize peer access latencies, to maximize query throughput and to balance the workload in the system. The object replicas can be used for load balancing and latency reduction.

2.2.7.3 Intelligent routing and peering

To realize the potential of P2P networks, it is important to understand and exploit the social interactions between the peers. The most pioneering work in studying the social connections among people is the "small-world phenomenon" initiated by Milgram [95]. The goal of his experiment was to find short chains of acquaintances linking pairs of people in the United States who did not know one another. Using booklets of postcards he discovered that Americans in the 1960s were, on average, about six acquaintances away from each other.

Adamic, et al. have explored the power-law distribution of the P2P networks, and have introduced local search strategies that use high-degree nodes and have costs that scale sub-linearly with the size of the network [96].

Ramanathan et al. [2001] determine "good" peers based on interest, and dynamically manipulate the connections between peers to guarantee that peers with a high degree of similar interests are connected closely. Establishing a good set of peers reduces the number of messages broadcast in the network and the number of peers that process a request before a result is found [97].

A number of academic systems, such as Oceanstore and Pastry, improve performance by proactively moving the data in the network (see also Section 2.1.1.6). The advantage of these approaches is that peers decide whom to contact and when to add/drop a connection based on **local** information only.

2.2.7.4 Security

Extensive research on this topic in P2P network has occurred because of requirements for anonymity[15], DRM [101], AAA [99], Firewall[103], and NAT [102].

The contemporary research such as Publius [115] addresses the encryption via public keys and multiple private keys as asymmetric manner. Recent improvements reduce the cost of Byzantine agreement, guiding future research regarding asynchronism in P2P network [99]. Anonymity of the receiving peer, sending peer and content creator shall be leveraged in a P2P system because a anonymity should be design in terms of business requirements described in chapter 2.

For DRM, we will not focus on this area because we focus open content. Open content distribution is free for everyone to use. However, we need to mention how P2P will be affected by the DRM.

Distributed computing P2P systems require execution of some code on peer machines. It is crucial to protect the peer machines from potentially malicious code and protect the code from a malicious peer machine. Protecting a peer machine typically involves enforcing (1) safety properties such that the external code will not crash the host, or will only access the host data in a type-safe way, and (2) enforcing security properties

[15] Please also refer to 2.2.4

to prevent sensitive data from being leaked to malicious parties. Techniques to enforce the include sandboxing, safe languages (e.g., java), virtual machines etc.

2.2.7.5 Digital Right Management

P2P file sharing makes file copying easy. It is necessary to protect the authors from having their intellectual property stolen. One way to handle this problem is to add a signature *in* the file that makes it recognizable (the signature remains attached to the file contents) although the file contents do not seem affected. This technique, referenced as *watermarking* or *steganography*[105], has been experimented with by RIAA [104] to protect audio files such as MP3s by hiding the copyright information in the file in **inaudible** ways.

2.2.7.6 Reputation

Free riders are a common problem in P2P file sharing systems. An user who only downloads files/objects, but not share the content with others. However, some of the users share lots of content with others. Therefore, one of the metrics is to identify whether a participant is "good" is their reputation. Accountability mechanism should be developed to identify peers with different reputations, for instance, one of approach is cross-rating. But ad hoc network have the possibility of unauthenticated and untrusted users. When multiple people are downloading the same file at the same time, they upload pieces of the file to each other. Not only does this redistribute the cost of downloading but it also eliminates free riding. Thus the node reputation can be resolved both regarding security and self-organization. In BitTorrent, there is a very neat strategy to resolve this free rider problem, via downloading while uploading. [4]

2.2.7.7 Accountability

This is another challenge in P2P, in terms of how to provide fair anonymity. Only if the overlay operator is able to know the identity of creator and delivery sites of specific content, can he ensure that the content creator and his or her content is traceable. This is not a technical problem, but rather about how anonymity is implemented and who the operator can trust to hold such information.

Firewalls and NAT are other challenges for P2P system. Since most Firewalls block TCP inbound traffic and many applications abuse port 80, which was only designed for HTTP, this causes communication difficulties between participants. If the participants are hidden behind different Firewalls communication becomes even harder. A common solution is configuring the Port used while installing the P2P software. But we need to look into this further and find out which method is used best. Maybe it is very rational to have more afford on the operator's side concerning other system features. For instance, a P2P relay/reflector server can be developed to find the peers if they are behind firewalls. We can also apply the same method for peers behind Network Address Translators.

As we described in section 2.1.1.2, every node can be assigned a node ID based upon hashing function. In this way, every peer can be addressed and indexed. This will efficiently overcome the problems caused by Firewall's and NATs. The key issue is how to map the key of the object and the key of the node to achieve best locality (i.e. minimal distance).

2.2.8 Transparency and Usability

There are many things which should be transparent in distributed systems. They included transparency of *location*, *access*, *concurrency*, *replication*, *failure*, *mobility*, *scaling*, etc. [104]. For example, in file sharing system, P2P users' localization experience will directly benefit from *naming/addressing transparency*. Besides this transparency, *administrative transparency* will deliver user-friendly configuration, and make user client upgrades more easy. Similar to interoperability, *network and device transparent* P2P systems will work on the Internet, intranets, and private networks, using high-speed or dial-up links. They should also be device transparent, which means they should work on a variety of devices, such as handheld personal digital assistants (PDAs), desktops, cell phones, and tablets. As an application layer overlay, it should be design to work on different devices such as desktops, laptops, PDAs, cell phones, TV-Set Top Boxes, and game stations. On the operator's side, automatic and *transparent authentication* of user and user agent can significantly reduce complexity from user perspective. Supporting mobile users to enable them to download files/objects independently no matter whether she or he is connected to the Internet. On the other hand, as a user, he can simply use a web interface rather than other protocols. Thus, naming/addressing, administrative transparency, network and device transparent, transparent authentication, mobility transparency can significantly enhance a P2P network's usability.

2.2.9 Fault-resilience

Decentralization eliminates a central point of failure, this has significant advantages. However, the solution for this problem varies when the system spans multiple hosts or networks. For example, it must consider disconnections/impeachability, partitions, and node failures. The problem is how to attach to those peers still connected in such presence of failures. How to connect to the remaining peers when the current neighbor disappears because a link is broken? How to resume the computation before the failure in order to continue the collaboration when connectivity is restarted? Perhaps, we should look into SETI@home and Genome@home[108]; as they attack this problem by partitioning the computations.

Another method to attack disconnection and node failure is a relay service. In the Groove[110] P2P system, they handle these problems by using some special nodes called *replays*, which store any updates or communications temporarily until the destination peer reappears in the network. Similarly, Magi [109] queues the messages at the source until the presence of the destination peer is detected. Yahoo[111] and ICQ[112] also adopted a similar approach for when peer is offline or disconnected.

Another problem is non-availability of a resource. The resource might not be available because of node failure, network link failure, because a node has gone offline etc. The popular method to resolve this problem is to replicate the resources or content. In this method, there are two approaches: passive and active replication. P2P systems such as Napster and Gnutella, implement both passive and an active uncontrolled replication mechanism based up on the file's popularity. It will be nice to provide persistent replication nodes to guarantee resource availability, but we need to look into the more active replication strategy and policy in that case. As I introduced in chapter 1, this can definitely exploit synergy from a CDN's replication strategies. In such a replication mechanism, we should not send replicas to those nodes which disappear or go offline or flip frequently all the time. Before sending replicas to a specific

replication node, a policy should be established to select good replication locations. Oceanstore has implemented such mechanism. The inspiration for such a policy can be borrowed from BGP's route flap dampening mechanism[113]. Thus a penalty can be added according to the history of certain destination peers which are candidates for the replication peer. During the time of transmission, a heartbeat message can be sent from the resource peer. Whenever there is no heartbeat from the resource, the receiving peer will notice the failure of the resource, then it will select a new replica. This method was used in [114]. Anonymous publishing systems such as Freenet and Publius ensure availability by controlled replication. Oceanstore maintains a two-layered hierarchy of replicas and by monitoring of administrative domains avoids sending replicas to locations with a highly correlated probability of failure. However, because a resource in the P2P system could be more than a just a file – such as a proxy to the Internet, shared storage space, or shared computing power – the concepts of replicated file systems have to be extended to cover additional types of resources. Grid computing solutions (e.g. Legion [167]) provide resilience against node failures by restarting computations on different nodes.

2.2.10 Manageability

A challenging aspect of P2P is that the system maintenance responsibility has been distributed completely and needs to be addressed by each peer to ensure availability. This is quite different from client-server systems, where availability is a server-side responsibility. The managerial issues include the following aspects:
(1) An operator of such network, must ensure that the upgrade system and maintain their system across. Adopting either a centralized management approach or self-managing approach will definitely effect decentralization, self-organization, fault resilient, and many other properties of the system.
(2) In file-sharing applications, P2P systems enable easy and fast retrieval of the data by distributing the data to caches located at the edges of the network. The location of the data is not known by the retriever, potentially even after the data is retrieved. Freenet, for example, stores the data in many locations in the path between the provider and the retriever, so the whole notion of hosting a file becomes meaningless. Files move freely among the peers and are allowed to disappear even if they are currently being downloaded. This has some important implications. For example, the question is who is accountable for the files (see Section 2.1.1.8.4). Also, how can we ensure that the entire data content is being downloaded, i.e., how do we cope with the un-reliability of the peers.
(3) P2P has the following implications for the IT industry: accountability, control, manageability, and standards. The first three are very closely related. Accountability is emphasized in centralized systems where access is monitored through logins, accounts, and the logging of activities. Accountability is more difficult to achieve in client-server systems, because of interactions with multiple clients. It is weakest in P2P systems, because of equal rights and distributed functionality among the peers. Similar reasoning applies for control. In centralized and client-server systems, control is done at one or more well-defined points, whereas it is harder to achieve in P2P systems, as control is entirely distributed. Therefore, more and more P2P systems adopt a hybrid structure for their P2P network.

2.2.11 Interoperability

According to earlier distributed systems research, the following are requirements for interoperability in P2P:

- The systems must interoperate.
- The protocols shall be used in systems communication, for instance HTTP, XML, SOAP, etc. must be in common.
- They must exchange signaling and traffics data across systems.
- The systems must determine if they are compatible at both the lower level and higher levels.
- Systems must advertise and maintain the same level of security , QoS, and reliability.
- A P2P system's protocols be ported into the other P2P system.

Interoperability means that the ability of systems, units, or forces to provide services to and accept services from other systems, units, or forces and to use the services so exchanged to enable them to operate effectively together [116]. There are many attempts for P2P system interpretability. One solution is to port a P2P middleware platform such as JXTA[117], BEEP[118], etc.; another is to build up a gateway to translate the protocols between each other[119]. In PlentyCast implementation, we can choose to develop our prototype based upon the JXTA platform as it is both open source and claims to be a de facto standard and nice adaptations exist for different OS platforms (including clear Java APIs for developers).

2.3 Core techniques

There are many core techniques, including location and routing, overlay construction, and security mechanisms. We only explain the former two in this paper.

2.3.1 Location and routing

In the following subsections, I will explain what most well known mechanisms of location and routing in P2P. They are: centralized directory model, flooding request model, and Distributed Hashing Table model. Then I will focus on introducing some outstanding Distributed Hash Table routing mechanisms.

2.3.1.1 Centralized directory model

This model was made popular by Napster and Audiogalaxy[120]. The peers of the community connect to a central directory where they publish information about the content they offer for sharing (see Figure 8). Upon request from a peer, the central index will match the request with the best peer in its directory that matches the request. The best peer could be the one that is cheapest, fastest, or the most available, depending on the user's needs. Then a file exchange will occur directly between the two peers. This model requires some managed infrastructure (the directory server), which hosts information about all participants in the community. This causes the model to have scalability limits, because it requires bigger servers when the number of requests increase, and larger storage when the number of users increase. However, Napster's experience showed that - except for legal issues – the model was relatively strong and efficient.

2.3.1.2 Flooding requests model

The flooding model is different from the central index one. This is a pure P2P model in which no advertisement of shared resources occurs. Instead, each request from a peer is *flooded* (broadcast) to directly connected peers, which themselves flood their neighbor peers etc., until the request is answered or a maximum number of flooding

steps (i.e. steps 2, 7, 6, 5) occur (see Figure 9). This model, which is used by Gnutella, requires a lot of network bandwidth for the discovery, and hence does not prove to be very scalable, but it is efficient in limited communities such as a company network.

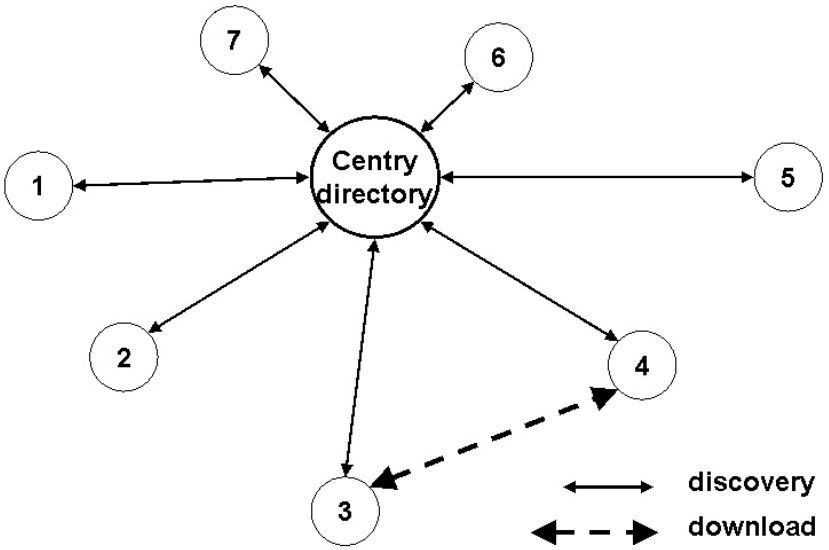


Figure 8. Centralized request model

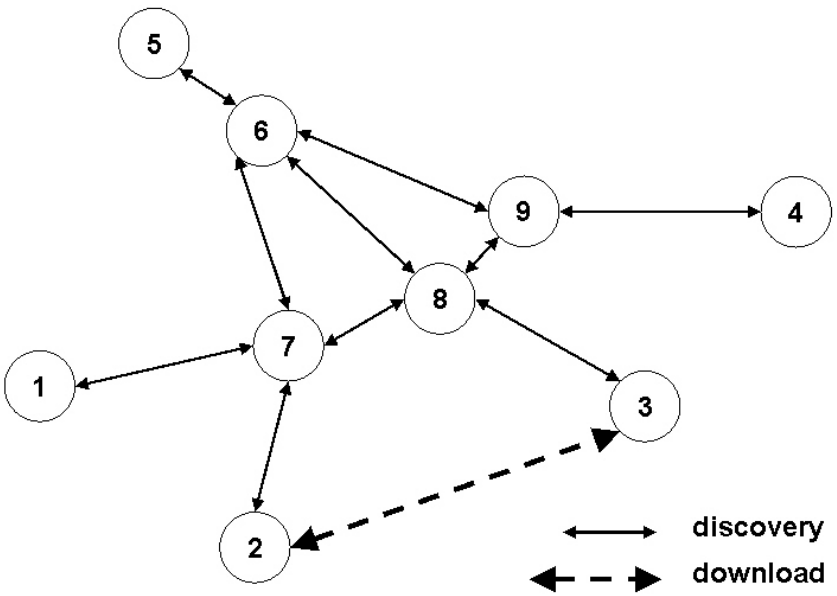


Figure 9. Flooding request model

To solve this problem, some companies have been developing “super-peer” client software, which concentrates lots of the requests. This leads to much lower network bandwidth consumption, at the expense of high CPU consumption. Caching of recent search requests is also used to improve scalability.

2.3.1.3 Distributed Hashing Table model

In response to the scaling problems exposed in the above centralized directory model and flooding request model, several research groups have (independently) proposed a new generation of scalable P2P systems that support a distributed hash table (DHT);

such as: Tapestry, Pastry, Chord, Content Addressable Networks (CAN), and Oceanstore. In these systems, utilizing Distributed Hash Tables (DHTs), files are

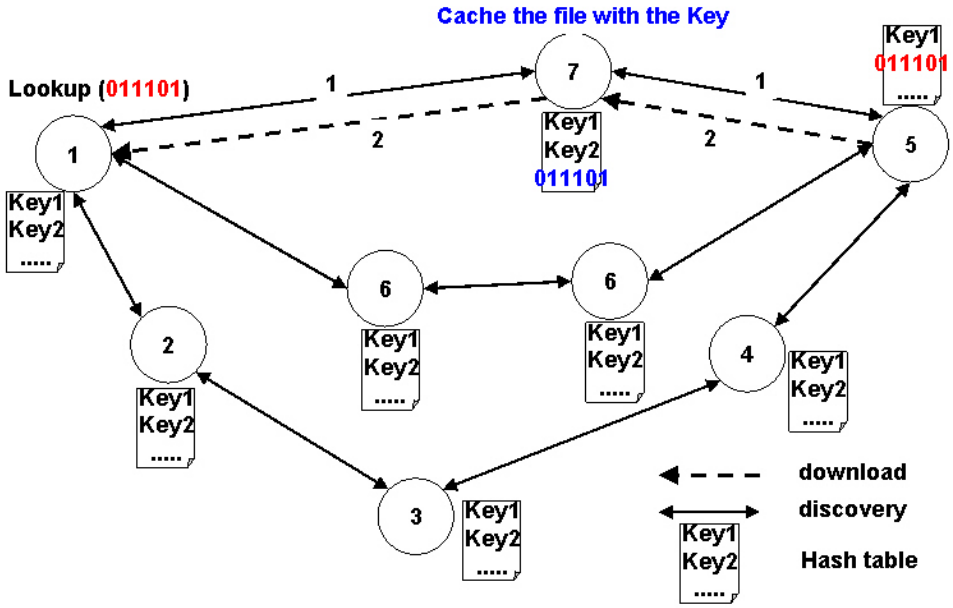


Figure 10. DHT Model

associated with a key (produced, for instance, by hashing the file name and its contents, or IP address and host name). Each peer from the network is assigned a random key and each peer also knows a given number of peers and each of them in the system is responsible for storing a certain range of keys. Each peer will then *route* the file towards the peer with the key that is most similar to the node key. This process is repeated until the nearest peer key is the current peer's key.

There is one basic operation in these DHT systems, lookup (key), which returns the identity (e.g., the IP address) of the node storing the object with that key. When a request originator issues lookup(key), the lookup is routed through the overlay network to the node responsible for that key. Then the document is transferred back to the request originator, while each peer participating in the routing will keep a local copy of the file. An example is shown in Figure 10. This operation allows nodes to put and get files based on their key, thereby supporting the hash-table-like interface. This DHT functionality has proved to be a useful in large distributed systems. However, it has the problem that the file keys must be known before posting a request for a given file. Hence it is more difficult to implement a search than in the flooding requests model. Also, network partitioning can lead to an *island* problem, where the community splits into independent sub-communities, that don't have links to each other. Therefore, we must find the best trade-off amongst geometry, distance, and algorithm of a specific DHT routing mechanism [121].

2.3.1.4 Plaxton location and routing

Plaxton et al.[122] developed what probably was the first routing algorithm that could be scalably used by DHTs. Although not intended for use in P2P systems, because it assumes a relatively static node population, it provides very efficient routing of lookups. The routing algorithm works by "correcting" a single digit at a time, for example: if node number 36278 receives a lookup query with key 36912, which

matches its first two digits, the routing algorithm forwards the query to a node that matches the first three digits (e.g., node 36955). To do this, a node needs to have, as neighbors, nodes that match each prefix of its own identifier, but differ in the next digit. For a system of n nodes, each node has on the order of O(*n*) neighbors. As one digit is corrected each time the query is forwarded, the routing path is at most O(*n*) overlay (or application-level) hops. This algorithm has the additional property that if the n² node-to-node latencies (or "distances" according to some metric) are known, the routing tables can be chosen to minimize the expected path latency and, moreover, the latency of the overlay path between two nodes is within a constant factor of the latency of the direct underlying network path.

The Plaxton location and routing system provides several desirable properties for both routing and location:

- Simple Fault Handling: Because routing only requires that nodes match a certain suffix, there is potential to route around any single link or server failure by choosing another node with a similar suffix.
- Scalable: It is inherently decentralized, and all routing is done by using locally available data in DHT. Without a point of centralization, the only possible bottleneck exists at the originate request node.
- Exploiting Locality: With a reasonably distributed namespace, resolving each additional digit of a suffix reduces the number of satisfying candidates by a factor of the ID base b (the number of nodes that satisfy a suffix with one more digit specified decreases geometrically). The path taken to the root node by the publisher or server S storing the object O and the path taken by client C will likely converge quickly, because the number of nodes to route to drops geometrically with each additional hop. Therefore, queries for local objects are likely to quickly run into a router with a pointer to the object's location.
- Proportional Route Distance: Plaxton has proven that the total network distance traveled by a message during both the location and the routing phase is proportional to the underlying network distance, assuring us that routing on the Plaxton overlay incuses a reasonable overhead.

There are, however, serious limitations to the original Plaxton scheme:

- Global Knowledge: To achieve a unique mapping between document identifiers and root nodes, the Plaxton scheme requires global knowledge at the time that the Plaxton mesh is constructed. This global knowledge greatly complicates the process of adding and removing nodes from the network.
- Root Node Vulnerability: As a location mechanism, the root node for an object is a single point of failure because it is the node that every client relies on to provide an object's location information. Whereas intermediate nodes in the location process are interchangeable, a corrupted or unreachable root node would make objects invisible to distant clients who do not meet any intermediate hops on their way to the root.
- Lack of Ability to Adapt: While the location mechanism exploits good locality, the Plaxton scheme lacks the ability to adapt to dynamic query patterns, such as distant hotspots. Correlated access patterns to objects are not exploited, and potential trouble spots are not corrected before they cause overload or cause congestion problems in a wide area. Similarly, the static nature of the Plaxton mesh means that insertions could only be handled by using global knowledge to re-compute the function for mapping objects to root nodes[123].

2.3.1.5 DHT algorithms benchmarking

In the following Table 1, we can find the difference between different DHT algorithms.

DHT-based P2P systems	Parameters	Hops to locate data	Routing path length	Notification on joins and leaves	Reliability	Overlay geometry
Chord	N- number of peers in the network	$\text{Log } N$	$\text{Log } N$	$(\text{Log } N)^2$	replicate data on multiple consecutive peers, appropriate retries on failure	Ring
CAN	d – number of dimensions N- number of peers in the network	$d \cdot N^{1/d}$	$2 \cdot d$	$2 \cdot d$	multiple peers responsible for each data item, appropriate retries on failure	Hyper-cube
Tapestry	N- number of peers in the network b- base of chosen identifier	$\text{Log}_b N$	$\text{Log}_b N$	$\text{Log } N$	replicate data across multiple peers, keep track of multiple paths to each peer	Tree + Ring (Hybrid)
Pastry	b- base of chosen identifier N- number of peers in the network	$\text{Log}_b N$	$b \cdot \text{Log}_b N + b$	$\text{Log } N$	replicate data across multiple peers, keep track of multiple paths to each peer	Tree + Ring (Hybrid)
Oceanstore	N- number of servers in the network (n-numbers of hashing functions i-size of the filter)[16]	$\text{Log } N$	$\text{Log } N$	$\text{Log } N$	Floating replica technique, prefetching and proactive migration, Bayou-like conflict resolution update technique	Tree

Table 1. DHT algorithms benchmarking

Five main algorithms have implemented the DHT routing model: Chord, CAN, Tapestry, Pastry, and Oceanstore[124]. The goals of each algorithm are similar. The primary goals are to reduce the number of P2P hops that must be taken to locate a document of interest and to reduce the amount of routing state that must be kept at each peer. Each of the five algorithms either guarantee logarithmic bounds with respect to the size of the peer community, or argue that logarithmic bounds can be achieved with high probability.

The differences in each approach are minimal, however each is more suitable for slightly different environments. In Chord, each peer keeps track of other peers (where N is the total number of peers in the community).When peer joins and leaves occur the highly optimized version of the algorithm will only need to notify other peers of the change. In CAN, each peer keeps track of only a small number of other peers. Only this set of peers is affected during insertion and deletion, making CAN more

[16] This will not be needed until the false positive rate will be tuned.

suitable for dynamic communities. However, the tradeoff in this case lies in the fact that the smaller the routing table of a CAN peer, the longer the length of searches. Tapestry and Pastry are very similar. The primary benefit of these algorithms over the other two is that they actively try to reduce the latency of each P2P hop in addition to reducing the number of hops taken during a search. In Table 1, I have learnt their benchmarking in according to studies [5], [123], [124].

The Chord algorithm models the identifier space as a one-dimensional, ring overlay geometry[121]. Peers are assigned IDs based on a hash on the IP address of the peer. When a peer joins the network, it contacts a gateway peer and routes toward its successor. The routing table at each peer *n* contains entries for other peers where the *i-th* peer succeeds *n* by at least. To route to another peer, the routing table at each hop is consulted and the message is forwarded toward the desired peer. When the successor of the new peer is found, the new peer takes responsibility for the set of documents that have identifiers less than or equal to its identifier and establishes its routing table. It then updates the routing state of all other peers in the network that are affected by the insertion. To increase the robustness of the algorithm, each document can be stored at some number of successive peers. Therefore, if a single peer fails, the network can be repaired and the document can be found at another peer.

CAN models the identifier space as n-dimensional. The overlay belongs to a type of hypercube structure. Each peer keeps track of its neighbors in each dimension. When a new peer joins the network, it randomly chooses a point in the identifier space and contacts the peer currently responsible for that point. The contacted peer splits the entire space for which it is responsible into two pieces and transfers responsibility of half to the new peer. The new peer also contacts all of the neighbors to update their routing entries. To increase the robustness of this algorithm, the entire identifier space can be replicated to create two or more “realities”. In each reality, each peer is responsible for a different set of information. Therefore, if a document cannot be found in one reality, a peer can use the routing information for a second reality to find the desired information. Thus it achieves quite good reliability.

Tapestry and Pastry are very similar and are based on the idea of a Plaxton mesh. Their overlay geometry is structured in ring and tree topology. Identifiers are assigned based on a hash on the IP address of each peer. When a peer joins the network, it contacts a gateway peer and routes toward the peer in the network with the ID that most closely matches its own ID. Routing state for the new peer is built by copying the routing state of the peers along the path toward the new peer's location. For a given peer n, its routing table will contain *i* levels where the *i-th* level contains references to *b* nodes (where *b* is the base of the identifier) that have identifiers that match *n* in the last *i* positions. Routing is based on a longest suffix protocol that selects the next hop to be the peer that has a suffix that matches the desired location in the greatest number of positions. Robustness in this protocol relies on the fact that at each hop, multiple nodes, and hence multiple paths, may be traversed.

OceanStore is based on Attenuated Bloom Filter and Plaxton mesh. Its overlay geometry is a tree structure. First to create a consistent distributed directory of objects and secondly to route object location requests. It uses a double routing mechanism: (1)Attenuated Bloom filters as the primary step; this allows the queried content to be retrieved efficiently with high probability; (2) then Plaxton routing whenever the first

algorithm fails. The first mechanism can fail because Bloom filters can sometimes be misleading owing to *false positives.* In Oceanstore the misleading behavior happens when attenuated Bloom filters indicate that two or more routes can lead to the object requested. This conflict cannot be avoided entirely, but it is possible to lower the probability of misleading behavior by choosing appropriate parameters for the Bloom such as number of hashing functions or filter size (width/bits of object ID). To allocate an object, if the local filter can't find it, then contact the neighbor for the first bit of that object key which possibly has a match. If the neighbor does not match, then forward the query to the next possible filter to process, and so forth till it match the key. As the inverse procedure of location, a replica can find its server in the replication phase of the object. Different from previous systems, when used as a network storage system, OceanStore uses floating replica strategy to replicate the objects and uses prefetching and proactive migration to achieve high reliability of shared data objects. To solve the problem in replica coherency across its overlay, it exploits similar mechanism to Bayou [125]. Also, it minimizes the costs in location and routing of logarithmic bounds with respect to the numbers of servers/peers.

2.3.2 Overlay network mapping

We argue that any solution that does not consider the underlying network structure will produces useless results for overlay networks. Fortunately, we have found many who people have the same thoughts as we have. The overlay network turns out to be an abstract set of hosts that play an active role in the data location process. As shown in Figure 11, the hosts in the overlay network are a subset of the total number of hosts present.

Note how end-to-end paths between two hosts can be different in the two layers. In particular a single edge in the overlay graph can include multiple edges in the corresponding underlying one. This lack of path correspondence between the two layers makes the routing process more difficult: a route retained as optimal in the overlay graph might not be optimal in the underlying graph. This problem is one of the most significant in the current implementations of DHTs. On the one hand, most designs take forwarding decisions at each hop, based on the neighborhood relationship in the overlay network. Depending on how the overlay network has been built, this can lead to awful results. For example, a node in KTH has its neighbor nodes in Europe and hence its path to a node in Stanford, California, USA may traverse distant nodes in Europe. Ideally, one would like to improve routing performance by avoiding such unnecessary high latency hops. Thus, a fundamental challenge in using large-scale overlay networks is to incorporate IP level topological information in the construction of the overlay to improve routing performance. This problem has been attracting more and more attention from researchers [123], [126], [127], [128], [129]. Basically, they look at this problem as a proximity problem. This includes two important steps in a overlay network: proximity routing and proximity neighbor selection.

2.3.2.1 Proximity routing

Proximity routing was first proposed in CAN. It involves no changes to routing table construction and maintenance because routing tables are built without taking network proximity into account. However, each node measures the Round Trip Time (RTT) to each neighbor (routing table entry) and forwards messages to the neighbor with the maximum ratio of progress in the d-dimensional space to RTT. As the

number of neighbors is small (2*d* on average) and neighbors are spread randomly over the network topology, the distance to the nearest neighbor is likely to be significantly

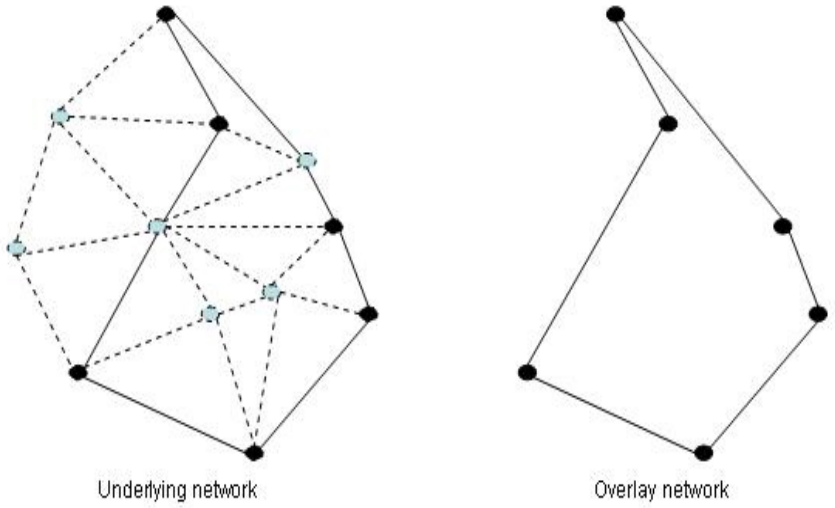


Figure 11. Overlay concept

larger than the distance to the nearest node in the overlay. Additionally, this approach trades off the number of hops in the path against the network distance traversed in each hop: it can ever increase the number of hops. Because of these limitations the technique is less efficient than geographical layout.

Proximity routing has also been used in a version of Chord. Here, a small number of nodes are maintained in each finger table entry rather than only one, and as I explained previously, a message is forwarded to the topologically closest node among those entries whose node ID is closer to the message's key. As all entries are chosen from a special region of the ID space, the expected topological distance to the nearest of the entries is likely to be much larger than the distance to the nearest node in the overlay. Furthermore, it appears that all these entries have to be maintained for this technique to be effective because not all entries can be used for all keys. This increases the overhead of node joins and the size of routing tables.

In [126], they proposed a binning strategy to solve this proximity problem.

Here, proximity routing offers some improvement in routing performance, but this improvement is limited by the fact that a small number of nodes sampled from special portions of the node ID space are not likely to be among the nodes that are closest in the network topology. For instance in the following Figure 13.

The rationale behind this scheme is that topologically close nodes are likely to have the same ordering and hence will belong to the same bin. They choose relative distances to achieve their “binning” strategy, *i.e.* latencies from this set of landmarks. A node measures its round-trip-time to each of these landmarks and orders the landmarks in order of increasing RTT. Thus, based on its delay measurements to the different landmarks, every node has an associated ordering of landmarks. This ordering represents the “bin” the node belongs to. However we can do better than just using the ordering to define a bin. They divide the range of possible latency values into a number of *levels*. For example, we might divide the range of possible latency values into 3 levels; level 0 for latencies in the range [0,80]ms, level 1 for latencies

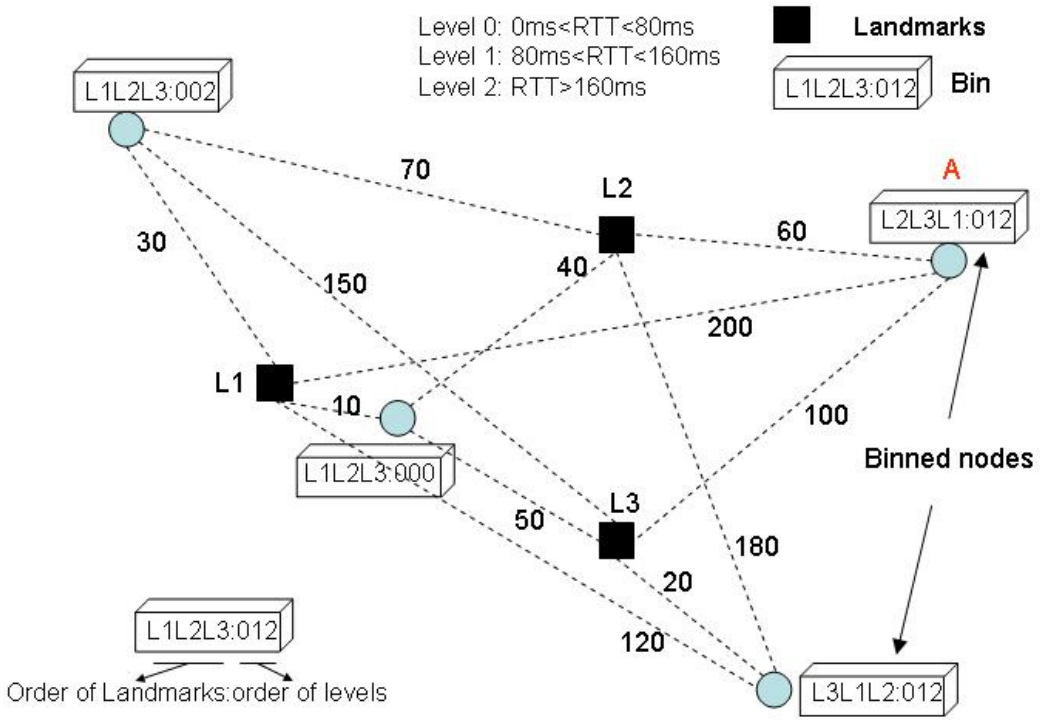


Figure 12. Binning Strategy concept

between [80, 160]ms and level 2 for latencies greater than 160ms. We then augment the landmark ordering of a node with a *level vector*; one level number corresponding to each landmark in the ordering. To illustrate, consider node A in Figure 1. Its distance to landmarks L1, L2 and L3 are 200ms, 60ms and 100ms respectively. Hence its ordering of landmarks is L2L3L1. Using the 3 levels defined above, node A's level vector corresponding to its ordering of landmarks is "0 1 2". Thus, node A's bin is "L2L3L1:012".

They find that the landmarks can achieve good scalability on making "bins" if there are 10 machines to be placed in the landmark farm. In according to their extensive simulation in power-law network topologies, their binning scheme does a reasonable job of placing nearby nodes into the same bin. This can be a good approach to achieve topology-aware node selection. Simply, the nearest node can be selected shall exist in the same bin or similar bins.

2.3.2.2 Proximity neighbor/server selection

The locality properties of Tapestry and Pastry derive from mechanisms to build routing tables that take network proximity into account. They attempt to minimize the distance according to the proximity metric, to each one of the nodes that appear in a node's routing table, subject to the constraints on node ID prefixes. Pastry uses the following metrics in its routing table:

1. Proximity invariant: Each entry in a node's routing table refers to a node that is near, according to the proximity metric, among all live Pastry nodes with the appropriate node ID prefix[17]. As a result of the proximity invariant, a message

[17] Pastry ID is 160 bits

is normally forwarded in each routing step to a nearby node, according to the proximity metric, among all nodes whose node ID shares a longer prefix with the key. Moreover, the expected distance traveled in each consecutive routing step increases exponentially, because the density of nodes decreases exponentially with the length of the prefix match. From this property, one can derive two distinct properties of Pastry with respect to network locality: distance traveled and route convergence.

2. Total distance traveled - The expected distance of the latest routing step tends to dominate the total distance traveled by a message. As a result, the average total distance traveled by a message exceeds the distance between source and destination node only by a small constant value.
3. Local route convergence - The paths of two Pastry messages sent from nearby nodes with identical keys tend to converge in the proximity space at a node near the source nodes. To see this, observe that in each consecutive routing step, the messages travel exponentially larger distances towards an exponentially shrinking set of nodes. Thus, the probability of route convergence increases in each step, even if earlier (smaller) routing steps have moved the messages farther apart. This result is of significance for caching applications layered on Pastry.

The routing algorithms in Pastry and Tapestry claim that they allow effective proximity neighbor selection because there is freedom to choose nearby routing table entries from among a large set of nodes. CAN also proposed a limited form of proximity neighbor selection in which several nodes are assigned to the same zone in the d-dimensional space. Each node periodically gets a list of the nodes in a neighboring zone and measures the RTT to each of them. The node with the lowest RTT[18] is chosen as the neighbor for that zone. This technique is less effective than those used in Tapestry and Pastry because each routing table entry is chosen from a small set of nodes.

In the distributed binning algorithm, sever selection process occurs as follows: (1) if there exist one or more servers within the same bin as the client, then the client is redirected to a random server from its own bin; (2)if no server exists within the same bin as the client, then an existing server is selected at random from the set of servers whose bin is most similar to the client's bin. The degree of similarity between two bins is defined to be the number of positions in their landmark orderings on which they match. There are three types of selection metrics that have been used, Hotz metric[130] (using inter-node distance calculated by node to landmark distance), and Cartesian distance(n-dimension of landmark number). Although the performance of Hotz distance based selection is competitive with the other two schemes, the researchers in [126] conclude that we really do not need to work very hard to achieve good server selection. Hence in designing such topology inference systems one might argue that the simplicity, scalability, and practicality of the system should be as important goals as prediction accuracy.

In practice, this server selection might be implemented by having the client include its bin information in a DNS query. DNS name servers could maintain the bin information for servers holding their content (for example, CNN's name server might

[18] RTT stands for Round Trip Time.

maintain the bin information for Web servers holding CNN content). Name servers might then use the above scheme to select a server for the requesting client.

2.4 Discussion

Equivalence is the essence of Pee-to-Peer technology. This enables different attributes of P2P in an P2P system such as decentralization, scalability, self-organization, anonymity, cost of ownership, ad-hoc connectivity, performance, security, anonymity, DRM, transparency and usability, fault-resilience, manageability, interoperability. File-sharing applications have shown excellence of scalability, decentralization, cost of ownership, transparency and usability. When we use this technology in our work, we shall take advantage of these attributes in order to achieve our project goals.

Chapter 3 Introduction to Swarm in content delivery

A latest performance study [11] shows that swarming[19] scales with offered load up to several orders of magnitude beyond what a basic web server can manage. Most impressively, swarming enables a web server to gracefully cope with a flash crowd, with minimal effect on client performance.

3.1 An overview of swarm in content delivery

The main constraint to scalable content delivery is the web's dependence on a client-server model, which is inherently limited in its ability to scale to large numbers of clients. As the load on a web server increases, it must either begin refusing clients or else all clients will suffer from long download times. This makes it difficult for a website with limited bandwidth to serve large files or a large number of requesting clients, particularly in a flash crowd event.

Many Peer-to-Peer systems have addressed this problem by using similar swarm techniques such as Gnutella, Swarmcast[145] and Onion Networks [144], BitTorrent[146], CoopNet [8], Pseudoserving [137], Backslash system[138], and PROOFS[139] and Swarming [11]. Basically, swarm intelligence[132] is one branch of Artificial Intelligence[131]. Swarm in content delivery is a peer-to-peer content delivery mechanism that utilizes parallel download among a mesh of cooperating peers. Comparing with content delivery in client-server paradigm, swarm create a mesh amongst the clients, the content data is transmitted in this mesh. Thus it significantly speed up downloading speed for a user, reduce content server load, and enhance content data availability. This is depicted in the following Figure. 14. In the following subsections, I will explain what key issues must be addressed in these swarm techniques.

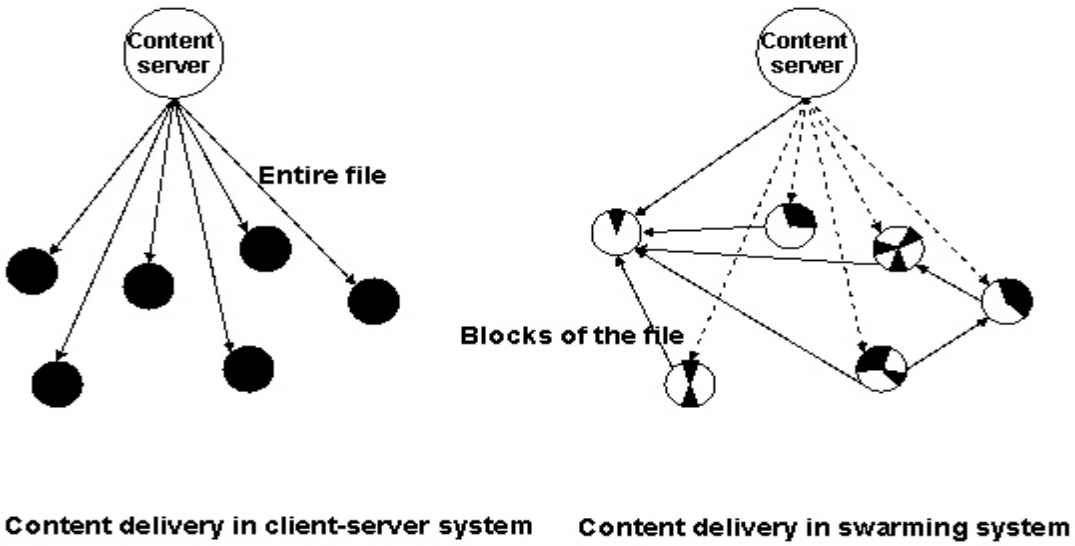


Figure 13. benchmark between client-server and peer-to-peer swarm in content delivery

[19] One latest proposal of swarm technique in study [11]

3.2 Core techniques in swarm content delivery

There are some commonalties amongst those swarm content delivery systems such as Gnutella, BitTorrent, CoopNet, Pseudoserving, Backslash system, and PROOFS and Swarming. First of all, they need to split the large file into small block/pieces. Secondly, they eventually form a full mesh amongst the peers who carries different percentage of a file or the complete file. Thirdly, in the mesh, every peer locate and download the block/s from another peer via certain cooperation protocol with each other. Some of them choose strategy of downloading and uploading simultaneously in one peer such as *Pareto Efficiency* in BitTorrent, and asymmetric mechanism in Swarming.

Before we go further deep into the techniques of swarm, I would like to examine the procedure of swarm in content delivery. In general, most swarm systems choose many-to-one and one to many transmission a model so that they can achieve many-to-many mesh. The key is how to form such a mesh for a given content. Figure 15 depicts the flow of swarm in content delivery. There are 6 phases should be very important in this work flow:

1. An new peer requests the content. In this phase, a new peer sends a request to the content directory for specific content downloading. Usually the content publishing becomes important. What information are important to the request peer/client in terms of the overall swarm strategy.
2. The content directory sends the new requesting peer with necessary information. These includes: list of the suitable siblings for this new peer to connect and partial content blocks, or just the peer list to the new peer who has the blocks of that content. In this phase, it is important for the new peer to select the right peers to download the content in terms of metrics such as distance, bandwidth, etc.
3. The new peer requests blocks from the peers on the list. In this phase, the new peer contacts the peers given by the content directory.
4. The new peer downloads from others concurrently. Many-to-one transmission has been established in this phase. The blocks of the content are sent from the one who has downloaded before to the new peer. During the transmission, fault resilient put important requirement for the request peer. How to deal with counterpart's disconnection or failure becomes very important for the download duration.
5. The new peer sends blocks to others. A full mesh for this content is created amongst peers. Now the one-to-many pattern has been established.. Thus a mesh amongst all downloading peers, mean while they should also upload. This mesh is created for a specific content. Due to the very high frequency of join and leave behavior in the mesh, how to deal with this dynamic situation become another important question to answer.
6. When the new peer complete downloading then leave the mesh. Another new peer and repeat (1) Except for the sequential join events, what if the system have multiple peers join simultaneously?

In the following sections, we will examine three typical swarm systems Swarmcast & Onion Networks[20], BitTorrent, and Swarming in terms of some key issues which happen during their content delivery. In Table 1, we summarizes different approaches to these key issues in the three typical P2P swarm systems.

[20] The people work in Onion networks are almost the same people worked in Swarmcast project before.

3.2.1 Splitting large files

In general, it is done by opening the large file and then read the conditions parameters for example block size or line numbers. The programme keeps reading the large file from STDIN and then writes to another temp file, if the condition is met then close the temp file and name or encrypt it using the naming policy. Then the read pointer moves to the next character in the original large file, continuously writing in a new file and so forth. When the end of the large file is reached the program is ended. For the detailed information please refer to Appendix 2. I also have tried some commercial products to split large file for example in Fast File Splitter [140]. They can provide more features such as encryption, see snapshoot in Appendix 1 for how it was done. There are many commercial file splitters on the market[141][142][143].

3.2.2 Initiated publishing

During replica placement before client access the content in CDN, a swarm delivery system also faces the challenge to make the correct placement of the content. There are two aspects that should be considered:

1. A content directory server needs to decide when to initiate (or stop) swarming for a particular file based on current server performance and the popularity of the file. Since it is not cost-effective to carry on swarming delivery when there are only a few users to access this content. Especially, if these users are very far away from each other.

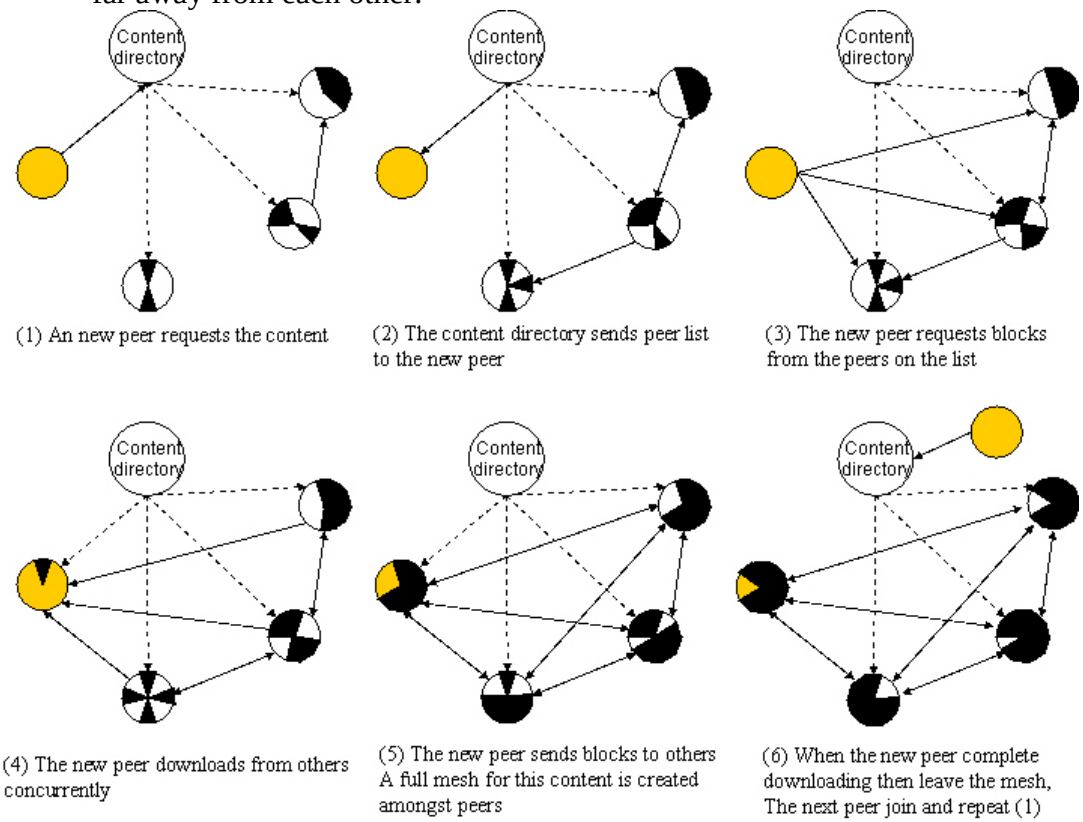


Figure 14. Swarming flow overview

2. While swarming happens, the content directory server needs to decide what portion of clients to send just a single block and how many to serve with the entire file. When a large number of clients access the content directory, server cost for sending replies is expensive. Depending on the strategy in 1, a careful consideration of what to send is needed. For instance, if you send too many

entire file copies to the clients, it may overload your server. In addition, if you send too few blocks to the clients, the swarm might not be formed effectively.

3. In such a dynamic hybrid P2P architecture, it is a challenge for the server to distribute the load to the clients via swarming delivery. The content directory server should trade-off between load-balancing of its own load and high availability of the content data in order to provide an effective swarming delivery. The goal of swarming delivery is to take advantage of high content availability to achieve fast download for user requests.

3.2.3 Mesh construction

To be able to deliver the content in many-to-one and one-to-many transmission model, it is the key to form and maintain a mesh connection in between the clients/peers who request the same content. This is the base for a peer to execute download concurrently of different blocks from peers, and upload what he has simultaneously. While doing swarming delivery, the client must deal with long-term dynamics and must determine when to add or drop a peer. In addition, because the client is using parallel download, it must decide which blocks to download from which server peer, while coping with the fact that each peer may potentially have a different set of blocks. Thereby, what protocols shall be used to communicate with each other in order to maintain the mesh. How much network status of other peers should be maintained when peers join and leave with very high frequency?

3.2.4 Peer and content identification

A client needs to locate other peers who have desired content so that it can use them as server peers. Whether a peer needs to be identified is a tricky question. If you devise a peer who should know the global status of the overlay network, identification is a must, for example in Chord, CAN, Pastry, and Tapestry. But if you don't want a peer have global status then you can ignore peer identification, for instance in Swarming delivery in [11]. The advantage of the former method is not only on the global status knowledge, but a good way of penetrate NAT or Firewalls because a peer ID can bootstrap the IP address of that peer which is usually hard to obtain behind NAT or Firewall.

Each content object must have a identification for many reasons. Firstly, because the block's identification replies on the content identification. Secondly, to avoid synchrony of the file name for different content. Thirdly, a unique ID of this content can avoid high costs of data coherency. Fourthly, it is of course to make content location easy. To maintain data integrity for blocks or pieces of the content, unique name of each block must be available.

3.2.5 Content/peer location

Once a client has located potential peers, it needs to decide which peers and how many peers it should use for parallel download. These are difficult choices because the client does not know ahead of time the average bandwidth available from each server peer. In addition, the bandwidth is changing all the time. In particular, the client does not know if the bottleneck of the connection will be local or remote. The optimal selection process involves the following metrics: distance, bandwidth, CPU load, memory usage, and number of processes in a peer.

3.2.6 Fault resiliency

The larger network is, the more frequent join and leave event will occur [93]. Due to the high frequency of join and leave events, in the middle of swarming delivery, it is possible that some peers leave gracefully (with notification) or ungracefully (without notification). This can prolong the downloading time of the entire file, when a uploading peer refuses to upload for their own reason. In this case, a re-selection process must occur. Moreover, this relates to how to exploit redundancy of the connections or peers to re-selection quickly. In addition, a strategy which makes download and upload happen simultaneously on one peer is desired to increase the probability of contributing to each node the mesh. This should ensure the peer will upload to the other in the mesh before it gets all the blocks/pieces. However, whether we shall choose a fairness principle should also be specified because more uploading peers naturally leads to shorter time to get the content file for each other. Moreover, how to adjust the downloading and upload connections for a peer can also be important to enhance the system performance.

3.2.7 End User bandwidth

From an access speed perspective, there are three types of users attached to the last mile of today's Internet: dialup user, broadband users, and office users.

	Downlink	Uplink
Dialup users	56Kbps	33Kbps
Broadband users	1536Kbps	128Kbps
Office users	43Mbps	43Mbps

Table 2. Class of Internet users

In the mesh, if every peer is downloading and uploading, those peers behind broadband modems will have a bottleneck on uploading their blocks to others because ADSL only provides asymmetric bandwidth for its users. In Table 2, a typical broadband user only has 128Kbps upload bandwidth, which is only 1/12 of their download bandwidth. The problem is that a single upstream pipe cannot meet demand of the another peer. Even worse, because of the fairness attribute of TCP's rate control [168], if the upstream path is congested, the downstream performance suffers as well. In another words, if a computer is serving files on the slow side of a link, it cannot easily download simultaneously on the fast side. How to detect ADSL peers and minimize the negative effect brought by asymmetry is one of the big challenges in swarming delivery.

Intuitively, dialup users can not upload very fast, therefore they can not offer very much download bandwidth for others. But their download bandwidth is not so different from their uplink. In the same mesh, we can imagine that the download duration will be longer if the peers are behind dialup modern. In another words, the broad band or office users will have shorter download time for a given large file. However, what if there is mixture of broadband users and dialup users and office users co-existing in the same mesh? Will the ADSL users be impacted by all other users in the mesh negatively?

3.2.8 ISP infrastructure

As I described in sections 2.2.3 and 2.3.2, an ISP's requirement and underlying network proximity should not be ignored in swarming delivery. In Figure 16, a traffic

generator simulated a Gnutella mesh. In such a mesh network, if there are many logical connections existing on one physical link, the physical link congestion will most likely occur. Moreover, if the node are not mapped (Figure 15) on the underlying network in an appropriate manner, the download duration for a file and efficiency of the mesh might be negatively impacted.

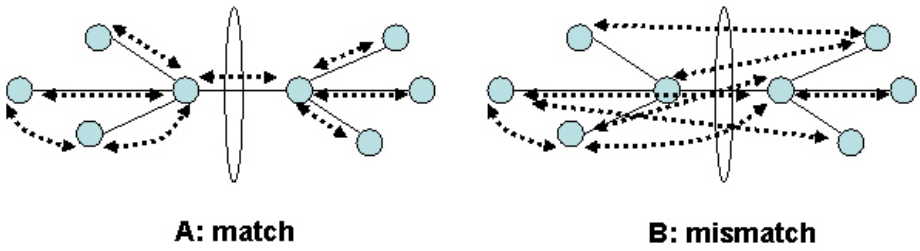


Figure 15. Match and mismatch in P2P overlay mapping

In addition, The larger the mismatch between the network infrastructure and the P2P application's virtual topology, the bigger the "stress" on the infrastructure[148]. This can increase the co-channel interference in physical media before the link gets completely congested. In according to my measurements, a large movie file downloaded via BitTorrent , from about 20 hops away from my machine. It is unlikely that traffic is localized in the mesh[21]. Most of the connections came across the peering links with Sprint.

	Swarmcast & Onion Networks	BitTorrent	Swarming
Initiated publishing	▪ Breaking large file into packets/pieces with FEC coding, then packets to be distributed randomly to the requesting clients ▪ Only distribute a portion of original packets ▪ An entire file must be downloaded in one of the clients before it gets Swarmcast in a mesh	▪ The .torrent contains information about the file, its length, name, and hashing information, and the URL of a tracker ▪ Starts a downloader from a node which already has the complete file (the 'origin')	▪ only distribute partial blocks of the file and a gossip[22] to all the requesting clients (conservative) ▪ root server ensures variety of blocks sent to the clients
Mesh construction	▪ A temporary mesh network for a specific file ▪ Node leaves the mesh when file reconstruction is successful ▪ Ignorant of the global state	▪ *Bencoding* formatted messages between the tracker and *metainfo* (.torrent) file ▪ BitTorrent peer protocol: downloaders periodically	▪ View this mesh as a *collaborative deliver system*, where server peers with larger portions of the file or higher bandwidth will tend to

[21] Please see Appendix 3

[22] A gossip message contains a list of peers that are willing to serve portions of the same file. For each server peer, the message lists the peer's IP address, a list of blocks the peer is known to have, and a time stamp indicating the freshness of this information.

		checking with the tracker to keep it informed of their progress, and are uploading to and downloading from each other via direct connections.	serve greater numbers of clients. Downstream peers will generally get pieces from upstream peers who have received the content earlier. ▪ During the mesh formation, individual blocks are propagated along a tree that starts at the root server ▪ server-based peer list together with gossip in the beginning ▪ gossiping in peer discovering
Peer/content identification	SHA-1 hash key for each piece	▪ A string of length 20 which this downloader uses as its id. Each downloader generates its own id at random at the start of a new download. This value will also almost certainly have to be changed if the same id exists. ▪ SHA-1 hash key for each piece	▪ No DHT identification for each peer, only IP address in case redirection overwhelm the root server due to large numbers of clients ▪ In the initiation stage, the root server supplies an initial set of peers
Content/peer Location	▪ Packets selection ▪ Prefer transfers from sources close together ▪ Closest distance(hops) priority	▪ Pieces selection ▪ Sub-piece priority ▪ Random first – when start to download ▪ Rarest first – replicate rarest pieces as soon as possible ▪ Endgame mode – last piece problem	▪ Priority on more blocks – the server peer who carries more blocks which the client need is first ▪ If the multiple peer provide the same content, then distance and bandwidth will be concerned
Fault resilient	▪ When a node receives a packet, it rebroadcasts it to other nodes; download and upload rate tend to be the same ▪ FEC creates additional repair packets ▪ Enable file re-construction with a subset of packets ▪ maximized and equal utility, i.e., every packet is equally important to the delivery of content ▪ The system requires that no explicit feedback be provided. A sender should not care what happens to a piece of data after sending it. This allows scalable transmission across broadcast or high latency/high error networks	▪ Tit-to-tat – downloading while uploading ▪ Choking algorithm – deal with uploading peer refusal ▪ Optimistic unchoking algorithm -deal with snubbing peer (uploading peer) ▪ Pipeline redundancy – deals with delay between sending pieces (by keeping 5 pipelines active at once and send subspecies on the five pipelines)	▪ Parallel downloading from multiple server peers ▪ Drop a server peer when it runs out of blocks, or disconnects ▪ Invoke peer selection process immediately when drop happens ▪ Select blocks from server peer or the content server ▪ No limitation of parallel connections
End User bandwidth	▪ Often, the sending side of a data transfer is the bottleneck for broadband users (i.e. , ADSL). To deal with this problem, given the current infrastructure of the Web, try to minimize the number of hops between the customer and the server. Swarmcast can do this through its aggressive mesh-	▪ Sometimes, limiting your upload rate will increase your download rate. This is especially true for asymmetric connections such as cable and ADSL, where the outbound bandwidth is much smaller than the inbound bandwidth. If you are see very high upload rates and low download rates, this is	▪ Results also demonstrate that broadband users do not see a significant performance increase when small numbers of office users participate in swarming ▪ Low-speed users will naturally decrease swarming performance for broadband users, but will not introduce

	creation policy. ▪ Swarmcast's simply sends out random packets, this allows even modem users to contribute to the system. So, a user on a cable modem could saturate his or her connection by downloading from a few dozen modem users in parallel. ▪ The only limit is an individual downloader's maximum bandwidth.	probably the case. The reason this happens is due to the nature of TCP/IP -- every packet received must be acknowledged with a small outbound packet. If the outbound link is saturated with BitTorrent data, the latency of these TCP/IP ACKs will rise, causing poor efficiency. ▪ BT chocking algorithm	significant problems
ISP infrastructure	▪ Quick scaling model localize the traffic from ISP to LAN, thus alleviate the bottlenecks on ISP networks.	No concerns on ISP network bottlenecks	No concerns on ISP network bottlenecks

Table 3. Benchmark of swarm systems

3.3 An introduction of Forward Error Correction codes

Error correcting codes or Forward Error Correction (FEC) techniques are a classical mechanism to protect the information against errors or losses in transmissions. The principle of FEC is to add redundancy to the transmitted information, so that it is still possible for receivers to recover all the whole transmitted data, even after experiencing errors in transmissions.

The main method of FEC used in the network domain is the use of block codes. These codes consider a group of k packets and compute *n-k* redundancy packets ($n>k$). The fundamental property of FEC permits the receiver to reconstruct the k source packets as soon as it correctly receives *k* packets amongst the *n* disseminated ones. This property enables very high reliability for data transmission, especially in a highly dynamic network.

FEC is used in several kinds of applications: wireless transmissions (e.g., satellite, cellular phone) data storage (e.g., CD-ROM, DVD.) or computer memory (e.g., RDRAM). In a networking context, FEC is classically used at lower layers (physical and link layer) in detection/correction mode. In higher layers, similar software techniques such as CRC or checksums are used to detect corrupted packets. In recent years, with the improvements of the performance of personal computers, it is possible to implement encoding and decoding of the FEC in software at user level (i.e., transport or application layer). For example, most reliable multicast transport protocols use FEC [150]. In data storage area, FEC is used to protect data against the failures of storage devices. Small scratches on CD-ROM are corrected by FEC directly implemented on the encoded information. Failures of hard disks can be protected by FEC-based systems such as RAID technology[151], [152]. This property holds only for maximum distance separable (MDS) codes, *e.g.* Reed-Solomon codes[149]**.**

As we know from previous sections in this chapter, large content data can be spit into fixed length small blocks and transmitted in a mesh. There are two very important attributes in such a mesh. One is that the join and leave behavior of each node is stochastic and it occurs at a very high frequency as we show in section 5.1.3. Another one is that the total volume of the data being transmitted in the mesh as relatively large, and sometimes can reach several orders of magnitude. Thus reliability of data

transmission becomes a big challenge in such mesh as created by swarming delivery. To achieve high reliability of swarming delivery, there are three fundamental ways of doing so.
First is to use Automatic ReQuest for retransmission. With ARQ, receivers use a back channel to the sender to send requests for retransmission of lost packets. ARQ works well for one-to-one reliable protocols, as evidenced by the pervasive success of TCP/IP. ARQ has also been an effective reliability tool for one-to-many reliability protocols, and in particular for some reliable IP multicast protocols. However, for one-to-very-many reliability protocols, ARQ has limitations, including the feedback implosion problem because many receivers are transmitting back to the sender, there is a need for a back channel to send these requests from the receiver. Another limitation is that receivers may experience different loss patterns of packets, and thus receivers may be delayed by retransmission of packets that other receivers have lost, but they have already received. This may also cause wasteful use of bandwidth to retransmit packets that have already been received by many of the receivers.

Second is the Data Carousel approach [153]. With Data Carousel, the sender partitions the object into equal length pieces of data, which we hereafter call source symbols, places them into packets, and then continually cycles through and sends these packets. Receivers continually receive packets until they have received a copy of each packet. Data Carousel has the advantage that it requires no back channel because there is no data that flows from receivers to the sender. However, Data Carousel also has limitations. For example, if a receiver loses a packet in one round of transmission it must wait an entire round before it has a chance to receive that packet again. This may also cause wasteful use of bandwidth, as the sender continually cycles through and transmits packets until no receiver is missing a packet [150].

The third is the FEC encoded approach. The rationale of FEC in swarming delivery is that the files become a set of $n+m$ blocks. Depending on FEC encoding technique, the first n blocks may or may not be the n source blocks. In first case, they just result from the original file splitting. The next m blocks integrate the redundancy introduced by the FEC encoding. In the second case, the original information contained in the n blocks is diffused into the $n+m$ blocks (viz. the source blocks can be in any i^{th} packets in $[0, n+m]$). When you publish them you just distribute various blocks over the mesh for instance as they have done as in swarming initiate stage in [11] . This distribution is ensured using a native service of the P2P architecture.

This dissemination algorithm can be based either on the natural dissemination due to the user downloads between the peers or on a more specific dissemination algorithm of the P2P system. Eventually, the distribution algorithm is similar to the approaches in section 1.2.4.2 because this is the same problem of placing the $n+m$ blocks amongst K peers. In the latter case, we shall consider distance, and bandwidth metrics for the dissemination because this will more accurately locate the cost for a peer in downloading. In addition, in study [11] , they find that when the various blocks are disseminated over the network, downloading a complete file is equivalent to downloading any n distinct blocks among the $n+m$ ones. As there are greater choices between the different blocks to get, the availability and the robustness of the system is increased. When these blocks are disseminated over the P2P network, the searching service helps to determine the closest ones by considering a certain cost function (e.g., the greatest bandwidth). Then, the n closest blocks are downloaded in a classical way. The original data file can be finally reconstructed by decoding of these n blocks. Moreover, they found that the following interesting figures:

In FEC Block Dissemination, for a 10 block file and an FEC encoding rate of ½ (i.e. 50% redundancy), each block is duplicated 5 times in 100 blocks. File downloading has an average cost per peer of **13.99**

Without FEC Block Dissemination, if there are 10 blocks in a file, them without FEC encoding, each block is duplicated 10 times in a total of 100 blocks. Thus file downloading average cost per peer of **17.03**

With entire file Replication, if there are 10 block in a file, no FEC encoding, each block is duplicated 10 times then in a total of 100 blocks. Thus file downloading average cost per peer is **16.85**[11]

Note that the cost function is purely based on the hops between each node.

Intuitively, we can see that the approach of block distribution with FEC encoding saves cost in a P2P overlay. Based on this, we shall choose FEC codes for the blocks. A good cost model shall consider metrics such as user downloading speed, bandwidth, server load, and network load. In addition, different download strategies such as fairness in downloading (downloading and uploading at once), and concurrent downloading.

Chapter 4 Introduction to Mobile Agents

4.1 A Definition

There are many definitions from academia and industry regarding Agent technology, because it is a relatively mature technology. I think the following definition can broadly define what an agent is:

*An **autonomous agent** is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future. It includes the following properties:*

Property	*Other names*	*Meaning*
reactive	*sensing and acting*	*responds in a timely fashion to changes in the environment*
autonomous		*exercises control over its own actions*
goal-oriented	*pro-active purposeful*	*does not simply act in response to the environment*
temporally continuous		*a continuously running process*
communicative	*socially able*	*communicates with other agents, perhaps including people*
learning	*adaptive*	*changes its behavior based on its previous experience*
mobile		*able to transport itself from one machine to another*
flexible		*actions are not scripted*
character		*believable "personality" and emotional state.*

It has the following taxonomy:

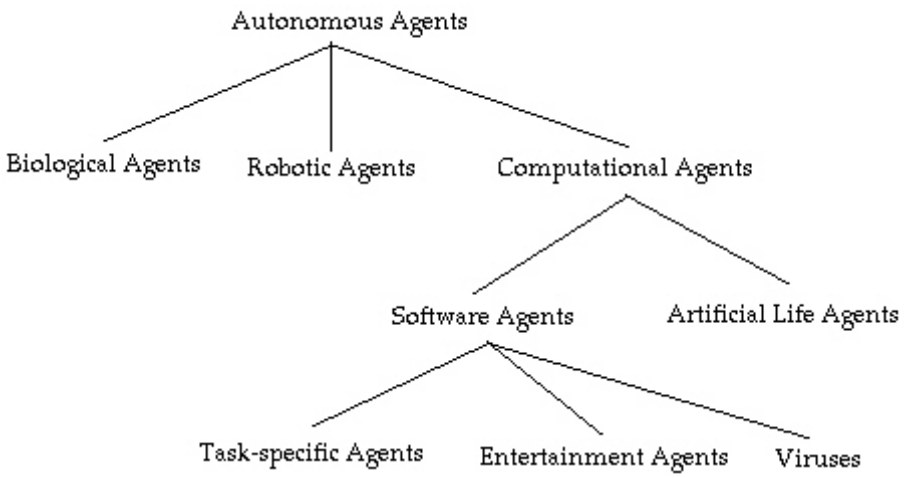


Figure 16. Agent Taxonomy [133]

Intuitively, the mobile agent belongs to one type of Task-specific Agents, and the property is able to transport itself from one machine to another. Therefore, we can accept this definition for a Mobile Agent:

"A mobile agent is a program that can migrate from host to host in a network of heterogeneous computer systems and fulfill a task specified by its owner. It works autonomously and communicates with other agents and host systems. During the self-initiated migration, the agent carries all its code and the complete execution state with it. Mobile agent systems build the environment in which mobile agents can exists. Migration of agents is based on an infrastructure that has to provide the necessary services in the network. The infrastructure is a set of agent servers that run on platforms (nodes) within a possibly heterogeneous network. Each agent server hides the vendor specific aspects of its host platform and offers standardized services to an agent that is docking on to such a server concluding migration. Services include access to local resources and applications, e.g. web-servers, the local exchange of information between agents via message passing, basic security services, creation of new agents, etc." [134]

4.2 What problems can mobile agents solve?

Agents are a tool for analyzing systems, not an absolute characterization that divides the world into agents and non-agents [135]. Agent technology has been integrated into many areas of computer science such as: objects and distributed object architectures, adaptive learning systems, artificial intelligence, expert systems, genetic algorithms, distributed processing, distributed algorithms, collaborative online social environments, security, etc. As mobile agent attributes mobility, it can be used to solve many problems in domains where agents can be integrated.

Agent technology solves, or promises to solve, several problems in different domains. Mobile agents solve the nagging client/server network bandwidth problem. Network bandwidth in a distributed application is a valuable resource. A transaction or query between a client and the server may require many round trips over the wire to complete. Each trip creates network traffic and consumes bandwidth. In a system with many clients and/or many transactions, the total bandwidth requirements may exceed available bandwidth, resulting in poor performance for the application as a whole. By creating an agent to handle the query or transaction, and sending the agent from the client to the server, network bandwidth consumption is reduced. So instead of intermediate results and information passing over the link, only the agent need to be sent. Here's a related situation. CDN is one of the typical examples as we described in chapter 1.

In the design of a traditional client/server architecture, the architect divides the roles of the client and server pieces very precisely – from the bottom to the up, at design time. The architect makes decisions about where a particular piece of functionality will reside based on network bandwidth constraints (remember the previous problem), network traffic, transaction volume, number of clients and servers, and many other factors. If these estimates are wrong, or the architect makes bad decisions, the performance of the application will suffer. Unfortunately, once the system has been built and the performance measured, it's often difficult or impossible to change the design and fix the problems. Architectures based on mobile agents are potentially much less effected by this problem. Fewer decisions must be made at design time, and the system is much more easily modified after it is built. Mobile agent architectures that support adaptive network load balancing could do much of the redesign automatically.

Agent architectures also solve the problems created by intermittent or unreliable network connections. In most network applications today, the network connection must be alive and healthy the entire time a transaction or query is taking place. If the connection goes down, the client often must start the transaction or query from the beginning if it can restart it at all. Agent technology allows a client to dispatch an agent handling a transaction or query into the network when the network connection is alive. The client can then go offline. The agent will handle the transaction or query on its own, and present the result back to the client when it re-establishes the connection.

Agent technology also attempts to solve (via adaptation, learning, and automation) the problem of getting a computer to do real thinking for us. It's a difficult problem. The artificial intelligence community has been battling these issues for two decades or more.

4.3 Core techniques in mobile agents

The key technique is how to migrate the program from the original host to its destination. Before we figure out how to migrate the agent, let's understand what network layers will be involved in the migration procedure shown in Figure 18.

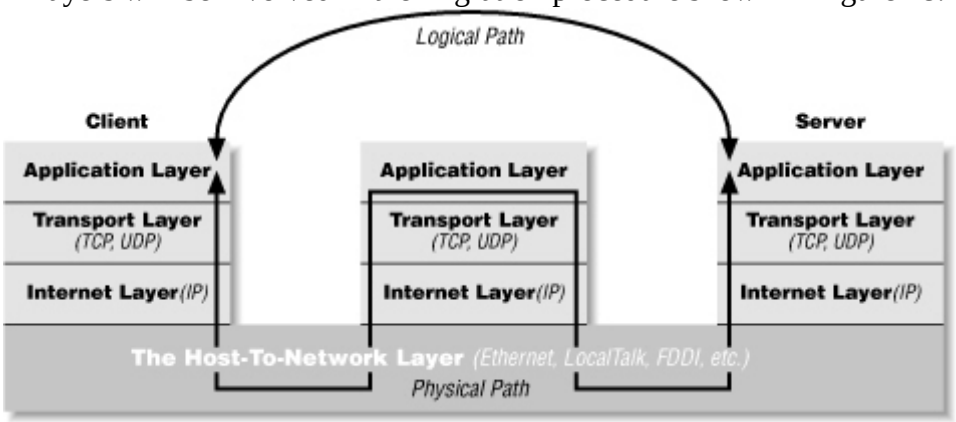


Figure 17. Network layers involved in migration

We assume that the physical path has been established between the client and the server in advance. The rest is to set up as a logical connection on the TCP or IP layer. Since the migration of the program happens on the application layer, there must be a communication channel between the client process and the server process, viz. a logical path must be established network connection between them on the TCP or IP layer according to RFC 793, RFC 791, RFC 919, RFC 922, and RFC 950. Fortunately, you don't have to do the work yourself. Sockets are an innovation of Berkeley Unix that allow the programmer to treat a network connection as just another stream into which bytes can be written and from which bytes can be read. Historically, sockets are an extension of one of Unix's most important ideas: that all I/O should look like file I/O to the programmer, whether you're working with a keyboard, a graphics display, a regular file, or a network connection. Sockets screen the programmer from low-level details of the network, such as media types, packet sizes, packet retransmission, network addresses, and more. This abstraction has proved to be immensely useful and has long since traveled from its origins in Berkeley Unix to all breeds of Unix, plus Windows, and the Macintosh. Thereby, to establish a network connection between the client and the server, you only need to do following the steps as follows:

1. Connect to a remote machine
2. Send data
3. Receive data
4. Close a connection
5. Bind to a port
6. Listen for incoming data
7. Accept connections from remote machines on the bound port

There are many programming language specific methods to interpret the above operations. For instance in Java, we have the following steps to establish a network connection:

1. The program creates a new socket with a Socket( ) constructor.
2. The socket attempts to connect to the remote host.
3. Once the connection is established, the local and remote hosts get input and output streams from the socket and use those streams to send data to each other. This connection is full-duplex; both hosts can send and receive data simultaneously. What the data means depends on the protocol; different commands are sent to an FTP server than to an HTTP server. There will normally be some agreed-upon hand-shaking followed by the transmission of data from one to the other.
4. When the transmission of data is complete, one or both sides close the connection. Some protocols, such as HTTP 1.0, require the connection to be closed after each request is serviced. Others, such as FTP, allow multiple requests to be processed in a single connection.

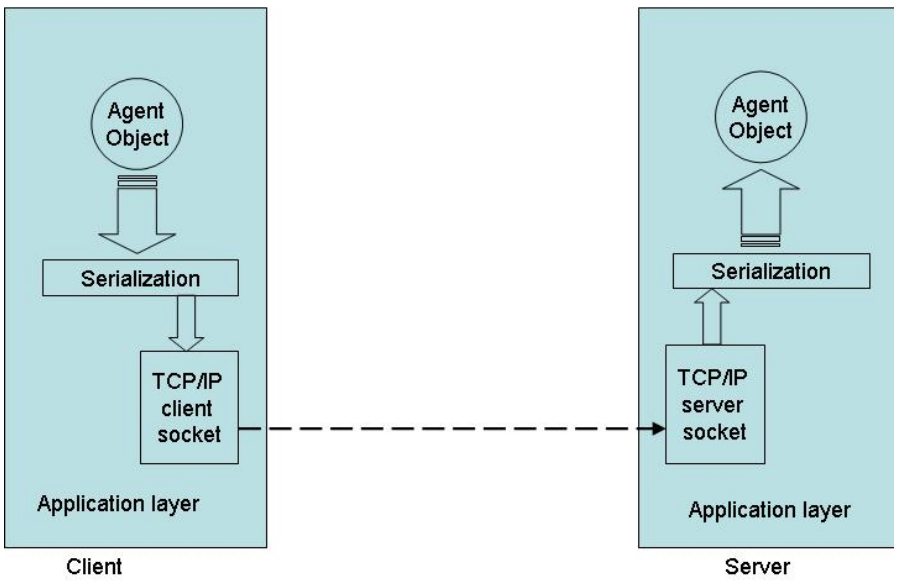


Figure 18. Migration implemented in Java

Given a network connection between the client process and server process, the next step is to wrap the program and send via this channel. There are also some programming language specific method to do so. We choose Java as an example again. In Java, statically, an agent is a type of classes which are defined as “extends Thread implements Serializable”. This means that this class is a child class of Thread class

(Java library default class) and it implements another default library interface class called “Serializable”. In the runtime environment, it can be a thread activated by a client or server process. When you instantiate a class like this, you get a serializable object for this class, then use the above 4 steps, you could send this object to the client or the server which un-marshal the object and get the member methods and output the content this agent carried as depicted in the following Figure 19.

4.4 Overview of the remaining chapters

As you can see from the previous chapters, there are many problems in CDN, P2P and Agents I have listed many solutions for these problems. However, what problems are we interested in? Why do we want to solve them? How to solve them? In the following chapter 4, I will explain our goals and use this list to bootstrap solutions to all the problems we are interested in solving. In chapter 6, we will propose a novel CDN architecture to solve the problems described in chapter 5. At the end of this paper in chapter 7, we will summarize our contribution and highlight the future work for further development because we believe that PlentyCast has a very good potential as a commercial application in the future.

Chapter 5 Problem statement

In this chapter, I begin by decomposing the five goals we set up for PlentyCast in this project. Secondly, I will describe the relationships amongst them and identify the high level problems that emerge. Third, I will describe the methods I used in this project to attack these problems and the criteria for how I select from different existing approaches to create an integrated solution.

5.1 PlentyCast design goals

In conventional CDNs, there are many problems needing further research in the following areas: cache placement/replacement, cache coherency, caching contents, user access pattern prediction, load balancing, proxy placement, dynamic data caching, etc [1]. Nowadays, new forms of content delivery such as video-on-demand, streaming content for real-time events, scalability, built-in security mechanisms etc, are major challenges for next generation CDN [2]. Since our aim is to deliver large static content in PlentyCast, our goals have been narrowed down so that we won't address all the outstanding problems of CDN listed above. In this project, we have selected the following goals in our PlentyCast system design; in order to achieve high system reliability comparing with other CDN design. The five goals are: improved access latency, improve network scalability, improve content availability, lower bandwidth consumption, and improve infrastructure performance compared to conventional CDNs. In the following sections, I will explain each of there goals in more details.

5.1.1 Improved access latency

A web user can *access* web content via a web application such as web a browser. Actually, it is not difficult to identify the two properties which are most important for the user when accessing content, the first is the *time* delay before content is visualized, and second is the *quality* of the content obtained. If we look at the access time, it is the *access latency* due to protocol stacks and the network connection between the web server and the user agent. The quality is determined by both the underlying network quality of service(such as specified for QoS in RFC 2211, RFC2212) and/or web application on both ends (i.e., the coding and decoding needed for specific types of content such as streaming audio or video content). Since we focus on static content in our paper, we assume the server has nearly infinite time in advance to encode the content. In another word, we assume the coding and decoding are well designed for all the content that PlentyCast will deliver, thus the issues of quality of content are excluded in this paper. In this section, we only look at latency problems. Latency is part of the CDN usability (as depicted in Figure 20.) because it represents one of the main user requirements in a CDN system.

As CDN delivers content from one end to another over the Internet, quality and latency are both important metrics to evaluate the CDN distribution/delivery quality of service. Latency for static content is determined by the following 5 aspects: (1) Number of requesting clients, (2) network distance between the client and the content server, (3) size of the content, (4) link capacity between the client and the content server, and (5) content server performance.

Intuitively, when more clients access the web server, more bandwidth will be consumed along the link, but the link capacity is a constant for a certain period of time; this could potentially lead to congestion along the link between clients and the server. For a more detailed description please refer to section 2.1.4.

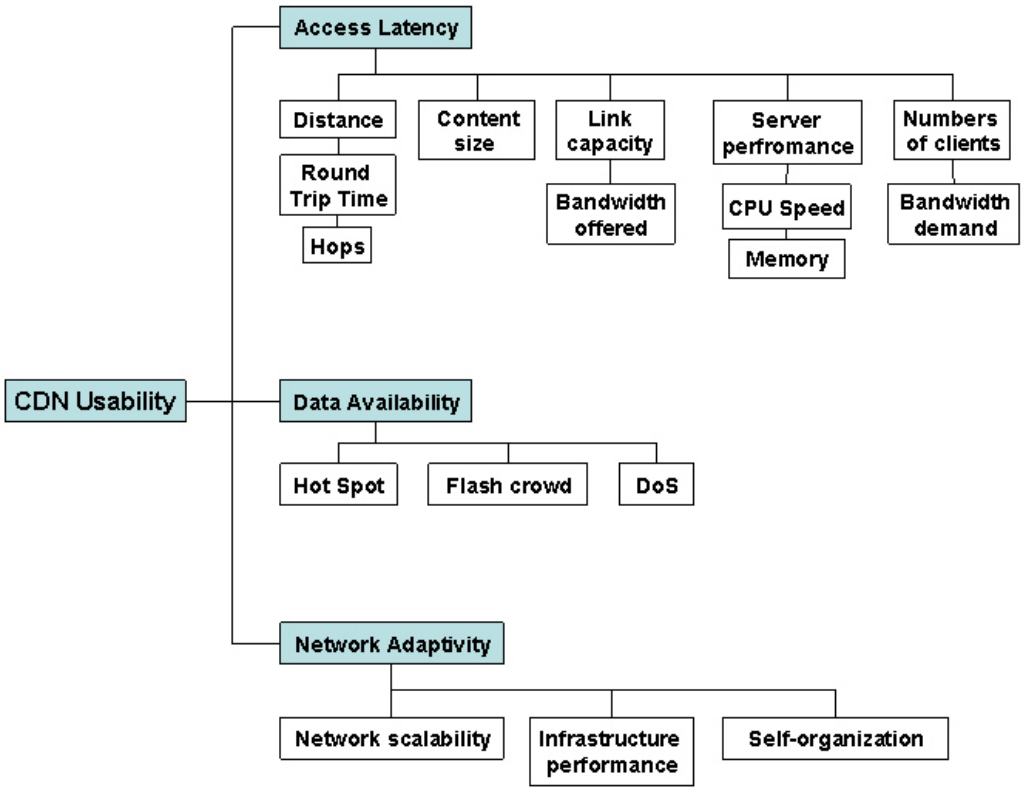


Figure 19 CDN usability decomposition

When every routing protocol is working well i.e. routing in LAN and external network are running normally, IP layer packets will traverse optimal (shortest path) routes. The number of network hops between the client node and server node could be one of the metrics to measure the latency. However, today's Border Gateway Protocol (an external routing protocol which is popular in today's Internet) appears to be quite instable due to many pathological routes residing in BGP routers [20]. Furthermore, this can hide the real numbers of hops between end systems. Some experiments also tell us that the numbers of AS –hops is an unreliable metric for indicating network latency [22]. Thus rather than using network hops to measure latency between the content server and its access clients, we choose RTT as our latency metric to indicate the latency. Thus if the system is well tuned, the file size directly determines the data transfer latency.

As we know from section 1.1.1, there are four Internet bottlenecks. However, physical link capacity is another aspect of the same problem and impacts the End-to-End transmission latency. Certainly when the client access rates do not exceed the link capacity, additional latency (i.e. beyond transmission latency) only relates to other factors. When bandwidth demand from clients exceeds the link capacity and/or the link capacity is reduced due to traffic shaping by ISP/s, congestion will result in "Page not found" errors to some clients or even worse behavior for some of the clients.

The content server used to be a very serious bottleneck before CDN was established. This is because content distribution directly relies on the server's CPU speed and memory capacity. It was quite easy to overwhelm a content server if there are more than hundreds of client accessing it concurrently. After CDNs were introduced, the CDN spreads the server load over its server farm (containing cache proxies and replica servers). This greatly alleviates the bottleneck due to the CPU and memory limitations of a content server. However, dynamic web content is increasing exponentially, and today's CDN have a problem, distributing the dynamic content. The most advanced technique is to let interaction happens between the content server and its access client as it is done in Edge Side Includes [9]. Thus, if there are a large number of users who request dynamic content, the problem of server performance reappears. Despite increasing hardware capacity in CPU and Memory, there are some additional ways to solve this problem such as using agent technologies in real time scheduling (however, we do not address this kind of problems in our project).

To sum up the access latency problems in CDNs, the following table lists the relations between each of these factors and latency.

Correlation between latency and its factors		Distance		Content data size	Link capacity	Content server performance metrics		Number of the clients
		RTT[23]	Hops		Bandwidth	CPU speed	Memory size	quantity
Measurement unit		Millisecond	number	Bytes	Bit/s	MHz	Bytes	number
Latency	longer	*longer*	*more*	*larger*	*Decrease*	*lower*	*smaller*	*larger*
	shorter	*shorter*	*less*	*smaller*	*Increase*	*higher*	*larger*	*smaller*

Table 4. Correlations between latency and its factors

Intuitively, every factor can change the latency independently. For instance, if there is larger number of client accessing the content server, this could lead to longer latency in the delivering the content to the user. The rest of the alternatives have the same property. If all factors accumulate products (the distance metrics in between are longer, content is larger, link capacity is smaller, content server performance metrics are relatively smaller, and the number of the clients is larger), then the latency will become very long.

However, these only reflect one aspect of this problem – how latency can occur. What we want to find out is the best trade-offs. How we can achieve improved access latency by altering some of these metrics is our focus.

Ultimately, latency is one of the most important metrics for web content access. Decreasing the latency between the content server and a user agent has become one of important mission of CDN. As I described in chapter 1, CDNs replicate the content close to (i.e., a short network distance) the user who requests specific content. This decreases the latency significantly by shortening the distance between clients and content servers. For instance, Akamai placed 13,000 servers across 1,000 networks in 63 countries [3]. In PlentyCast, we adopt the same approaches to resolve the latency problem, but we expect to reduce it more than conventional approaches have.

[23] RTT: Round Trip Time, in the traceroute command, is the time length for a packet to traverse back and forth between destination and source.

Most conventional CDNs decrease latency by two techniques, one is to intelligently place the content via replication near the requesting user (shortening distance between content and user); another is to intelligently locate and route the content data to the requesting user. These are certainly correct solutions to reduce latency. However, we introduce a third technique to improve access latency in PlentyCast – this is swarm delivery. Basically the mechanism is to partition large content into small fragments; these small pieces of content can be distributed near the requesting user. When a user's access is authorized, all these smaller fragments will be transmitted from **different** locations to the destination (forming a multi-point-to-single-point traffic pattern). At the destination, the user agent receives all the fragments concurrently, and assembles them to restore a complete copy of the content at the destination. This appears to be very effective in reducing the latency for a client to access large content data as demonstrated by some new Peer-to-Peer software such as BitTorrent [4]. Thus we argue that the existing CDN techniques with the addition of swarm delivery technique can deal with large content. This enables PlentyCast to reduce access latency better than the approaches used in current CDN networks.

5.1.2 Improve network scalability

Scalability means the ability of a network system to increase numbers of servers and clients smoothly. In general, the more clients a CDN can serve and the better self-organization when clients leave and join at a high rate, the better scalability a CDN has. Nowadays, most of CDN systems are designed following the Client-Server paradigm. As we know, this centralized system pattern exposes synchronization and coordination problems when the network becomes very large or topology change becomes highly dynamic. As a new type of computation paradigm, Peer-to-Peer demonstrates excellent scalability and has great potential for content distribution in very large scale networks. An immediate benefit of decentralization is improved scalability [5]. File sharing applications are good examples, such as Freenet [42], Gnutella [40], Kazza [41], and BitTorrent. There have been many attempts to discover how to use P2P in CDN [6], [7], [8]. We can use P2P technology in a CDN is to take advantage of clients' CPU, storage, and memory for content distribution in certain circumstances. Intuitively, if we choose a client as part of the CDN, content data availability can be significantly increased because the storage space has been extensively expanded in this case. PlentyCast can scale large numbers of clients to access the content comparing traditional CDNs, particularly when there is some extreme events happening in the content delivery network, such as in the case of "Flash Crowds" and Delay of Service attacking on any content server (see section 5.1.3). In PlentyCast, content location and routing, and self-organization must be considered when we want to achieve this goal. Moreover, content distribution, selection, and content lookup algorithms are key issues. This is what we want to focus on in the reminder of the report, in order to achieve better scalability than conventional CDNs.

5.1.3 Improve content availability

Content availability means that content data is available for any requesting user at any point in time over the network. In general, it is achieved mainly through redundancy involving increasing the number of places where the content data is stored, hence more nodes where it can be reached. Intuitively, replicating content data in a replica server farm is one of the major techniques to achieve high content data availability in a CDN. Nowadays, one of the key technologies of conventional CDN is Edge Side

Includes [9]. This technology enables us to cache content or replicate content as close to the user as possible. Today's Internet has been partitioned into thousands of Autonomous Systems, all of them are connected via the Border Gateway Protocol over physical links. Users are sitting at the leaf nodes included in each AS. Within an AS, if we place the content cache or replicate the content in the CDN server at the edge of this AS, it will significantly shorten the distance metrics between the content and any content user (or potential user) within this AS. When a user attempts to access the original content server, he or she doesn't access the content from the original content server, but instead receives the content from the replica server or cache proxy at the edge of the same AS as the user node is located. In this way, on one hand, latency has been significantly decreased between the content data and the user. On the other hand, data replication/redundancy increases content availability. However how much is data availability improved by such technology?

9/11[24] gave us a comprehensive lesson about "Flash Crowds"! After the world trade center collapsed on Sept 11 2001, thousands of Internet users accessed MSNBC (who was a customer of one of biggest CDN operators) web servers in very short time – this is a so called Flash Crowd. This behavior makes the web site unreachable because the servers are overwhelmed. When this has been done maliciously, the consequence is recognized as a Denial of Service attack. The following was documented by Microsoft's CoopNet project during 9/11:

> *We use traces collected at MSNBC during the flash crowd of Sep 11, 2001 for our evaluation. The flash crowd started at around 1:00 pm GMT (9:00 am EDT) and persisted for the rest of the day. The peak request rate was three orders of magnitudes more than the average. We report simulation results for the beginning of the flash crowd, between 1:00 pm to 3:00 pm GMT. There were over 300,000 requests during the 2-hour period. However, only 6% or 18,000 requests were successfully served at an average rate of 20 Mbps with a mean session duration of 20 minutes. Unsuccessful requests were not used in the analysis because of the lack of content byte range and session duration information.*[10]

We think this problem is not simply due to CDN scalability. Ultimately, this is a problem of content data availability. The lessons to be learn from this event are not only how to deal with "Denial of Service" attack, but more importantly, how to make data highly availability for certain large size of instances of interesting content while protecting the content server from being overwhelmed by a Flash crowd.

Recently, the NASA[25] spacecraft Spirit landed on Mars but almost swamped NASA Web site due to 109 Million Hits in 24 Hours [63]. *"To handle the Mars traffic, NASA is paying $1.5 million to eTouch Systems Corp., Speedera Networks Inc., and Sprint Corp. beyond the $3.5 million it pays them each year to handle NASA's more popular Web sites, said Jeanne Holm, NASA's Web portal manager."*[19]. This reflect a fact that conventional CDNs can only deal with a flash crowd by either interconnecting CDNs (solution such as CDI) and/or increasing the replica servers and/or cache

[24] http://www.cnn.com/2001/US/09/11/chronology.attack/ accessed on 2004-01-18 16:40
[25] http://www.nasa.gov/externalflash/sts107/index.html accessed on 2004-02-07 16:14

proxies at high expensive. As our best knowledge, there are other solution to enhance content availability for content providers.

Despite technical difficulties, there is another important fact we should not ignore – lack of good content. Why do very large numbers of users go to NASA, CNN, MSNBC, and The Wall Street Journal on line, NASDAQ for reading or watching news? This lack of quality content occurs because on Internet causes large numbers of users to access relatively small numbers of content servers. This will likely to last for a relatively long period in the future, especially for the large-size content. Hence traffic hot spots are easily formed. Enhancement of data availability is the key to have this subset of content distributed to relatively large numbers of Internet users.

Differing from Edge Side Includes [9] technology and CDI [67], we argue that P2P content caching should not be restricted to CDN server farms, but should use the many clients' storage too. This can create a much larger storage space enabling higher data availability. By introducing swarm delivery in PlentyCast, data can be more widely distributed and more quickly located and transmitted to the destination. Further more, a downloading while uploading strategy shall be adopted in swarm delivery. Originally, this strategy was adopted to address the problems of free-riding [4] in Peer-to-Peer file-sharing applications. Additionally, we use it for a third purpose: gracefully avoiding Flash Crowd and counteracting Denial of Service attack. As downloading comes from the other clients uploading, the more clients that request the same content the more content data will be distributed thus relieving original content server and CDN replica servers. A disadvantage of swarm delivery is that it may occupy too many ports and steal too much bandwidth from other (normal) client applications [11].

Ultimately, realizing swarm delivery and download while uploading strategy relies on the careful design on algorithms in content distribution, location and routing systems of PlentyCast.

5.1.4 Lower bandwidth consumption

As we described in paragraph 1.1.1, there are four bottlenecks existing in today's Internet. Based upon our understanding of the existing approaches to these four classes of Internet bottlenecks[26], we will describe our solutions for alleviating these bottlenecks in our network.

Many people think Peer-to-Peer file sharing applications save lots of bandwidth due to its equivalence property, i.e., peer A can retransmit the content to peer B thus saving bandwidth between the peer B and the content server. My viewpoint is that this only saves the bandwidth on the first-mile (as CDN does), but indirectly puts increased load on peering and backbone links unless the peers are carefully chosen. Given the pressure from broadband DSL subscribers, I don't think peering and backbone/infrastructure traffic engineering is an easy mission. Additionally, P2P users usually consume more bandwidth then normal users. Imagine how much traffic 230 million Kazza users will generate on the Internet. According to an industry estimate:

"*P2P now accounts for 50 to 70 percent of all Internet traffic. KaZaA alone has more than 230 million client downloads, with about 900 million files available for sharing, representing about 6,500 terabytes"* [14].

[26] Please refer to 1.1.1

Ultimately this expansion in bandwidth is inevitable because of layer seven services are becoming more and more diversified and more and more bandwidth intensive applications have been unleashed. Reducing bandwidth consumption is difficult in a conventional CDN, but it will be possible by using PlentyCast. We use an intelligent distribution strategy of CDN to alleviate the first-mile bottleneck, P2P solves the last-mile bottleneck, and self-organization to solves the bottlenecks for peering and infrastructure/backbone. Therefore, we can reduce the bandwidth consumption at each of the bottlenecks in our CDN, this potentially leads to reducing the bandwidth required over the entire Internet.

5.1.5 Improve infrastructure performance

As we know, traffic engineering is the key task for ISPs to avoid network performance and quality of service problems. How to achieve reasonable network performance is not the only question when we construct an overlay on top of the ISP's layer 3 network. As a matter of fact, new types of Internet traffic models have been discovered by academic researchers and so many new types of service applications are unleashed. This is reflected in more and more new types of traffic patterns forming. ISP's traffic engineering and planning becomes more and more difficult because of highly dynamic traffic patterns appearing in relatively short periods of time. For instance, routers/switches upgrading for adapting P2P traffic control (as P-cube [15] does) lags the utilization of new P2P applications [18]. On one hand, find solutions on the underlying network is interesting since this layer is supposed to guarantee QoS on layer 3. But the news is not optimistic, skewed traffic of P2P over large networks [17] reflects 20% of users generating 80% of Internet traffic. The consequence is the other web application's bandwidth has been significantly consumed. Even though an administrator can shut down P2P systems on a campus [158], shutting down P2P will result in fast customer churn in an ISP. These dilemmas suggest that a new way of thinking about how to resolve this traffic engineering bottleneck in order to maximize network performance. This is not only for P2P traffic, but also for all the new types of layer seven traffic that will be generated. This should be the way to handle the bandwidth intensive applications, otherwise, it will not exist for very long. For instance. P2P systems will not exist any longer if they have both annoy ISPs and the record industry (a class of Content Provider). Therefore, we must find harmonized solutions for all different types applications in today and tomorrow's Internet. This means that the solution must consider all the requirements from different actors in the networks

This goal will make the PlentyCast design distinct from many other CDNs. When we attack this problem, there is a very important philosophical motivation we advocate:

The survival of the fittest

– Charles Robert Darwin

If we examine the problem of Internet infrastructure performance, I believe that coadaptation is the key to solving this type of problem. This means that a solution providing solutions for both infrastructure and applications make a good solution for everyone. However, it is naïve to think that every P2P system devised will be infrastructure-aware since they have been vastly used to do decentralized file sharing. Thus we choose to describe our PlentyCast approach because a CDN is more manageable than P2P file sharing systems. As one of the pillars of PlentyCast, self-organization will be used to provide fault tolerance for the overlay network. In addition, PlentyCast's self-organization means that system can re-organize overlay traffic to optimize underlying network's performance. This will enable our CDN to

adapt much better to the underlying network. Ultimately, this maximizes the underlying network performance for content distribution. To our best knowledge, this is the first study this approach in CDN, we will refer to it as a **Both Infrastructure and Overlay Network Based Solution**. **(BIONBS)**

5.2 Problem modeling

Our goals in designing PlentyCast was to realize our goals by exploiting the following systems features – overlay self-organization, intelligent content location and routing, and swarm delivery. All the goals are inter-related, and this is depicted in Figure 20.

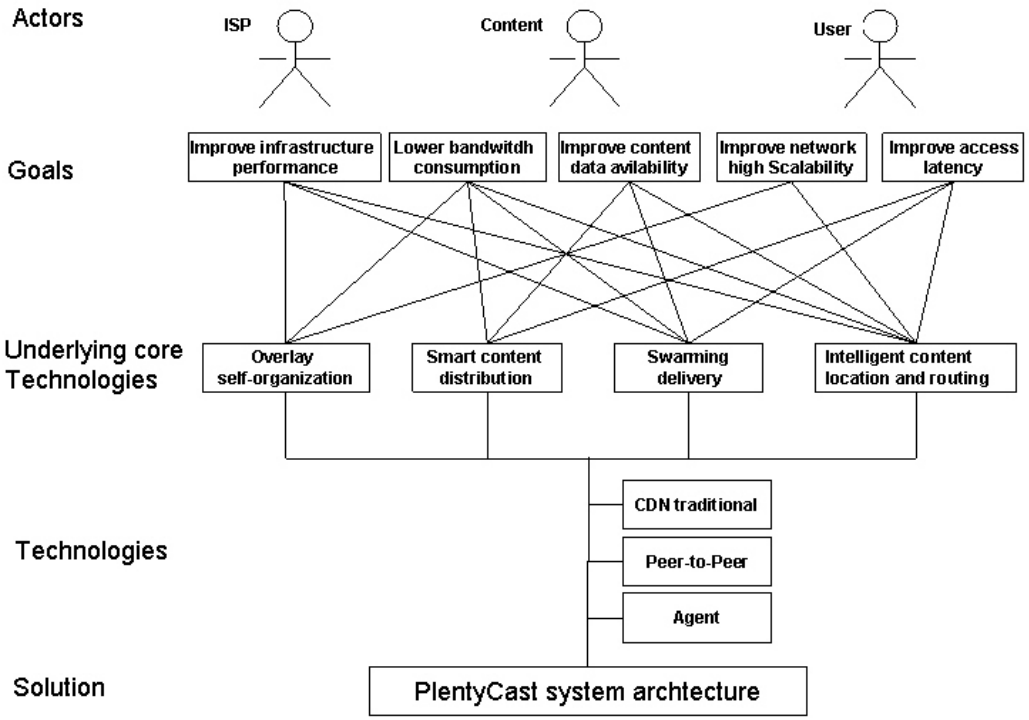


Figure 20. Problem model

In this picture, the questions are concentrated on the layer underlying. The following questions should be asked:

A. What functions shall we design in to use each core technology?
B. How to design these functions?

To answer these two questions, we must answer the following questions:

1. What requirements shall we consider from a user perspective?
2. What requirements shall we consider from a content provider perspective?
3. What requirements shall we consider from an ISP (multi-tier) perspective?
4. What technologies shall we choose to realize those requirements?

Following this, we can define a set of sub-problems for each of the core technologies:
Smart replica placement:

1) How to identify each content replica?
2) How to place content as close to users as possible?
3) How many replicas of the content shall be placed close to end users?
4) How to replace content in terms of self-organization?

Intelligent replica location and routing

1) How to locate a replica in the content space?

2) How to choose the best replica to download in terms of location, routing, and self-organization metrics?
3) How to relocate new replica on downloading?
4) How many replicas shall be chosen to download?
5) How to organize content and its host/s efficiently in a dynamic environment?
6) How to compute the location and routing metrics?
7) How to acquire the underlying traffic engineering metrics?

Swarming delivery

1) What information shall be carried in metadata for each fragment of content?
2) How to document each metadata is host?
3) How to ensure integrity of metadata during transmission?
4) How to aware the other pieces of the same content?
5) How to split and rebuild the content locally?
6) How to avoid expensive negotiation between hosts in swarming?

Self-organization

1) How to detect faults during content transportation?
2) How to isolate the faulty nodes?
3) How to signal location and routing to relocate content following faults?
4) How to re-organize overlay traffic in terms of metrics from infrastructure?

5.3 Discussion of the criteria

According to distributed system theory, there are many aspects that should be addressed in such a large system design. However, due to the limited time and resources in this thesis project, I will focus only on major problems addressed in this system architecture. We have chosen to focus on self-organization, replica placement, swarm intelligence, and location & routing after intensive literature study. We believe those four technologies are most outstanding candidates for solutions to the problems to be addressed by this design. These mechanisms will form backbones of PlentyCast. There are many contemporary researches addressing these four areas. In replica placement, swarm intelligence, and location & routing, there is no major difference between our approaches and others. With regard to self-organization, we consider that infrastructure awareness is one of the important system metrics. This makes a big difference from the other approaches.

In chapters 3, 4, 5 and 6, we first decompose the problem domain into sub-problems. Following, a study of relevant literature from leading industrial vendor's research laboratories such as HP lab, IBM research, Microsoft Research, ATT Research, and standardization bodies such as ACM, IEEE, and IETF. Third, I looked at the academic research journals by following the citations in many papers.

As we know, end users, content providers, and ISPs are the three major customers of CDNs. A good CDN system should not be skewed to one of these customers over others.

In this paper, we focus on mechanisms and/or algorithms. No simulations were conducted, but we believe this should be part of our future work. An industry team will implement prototype of this design following my thesis project.

Chapter 6 A novel architecture – PlentyCast

6.1 System overview

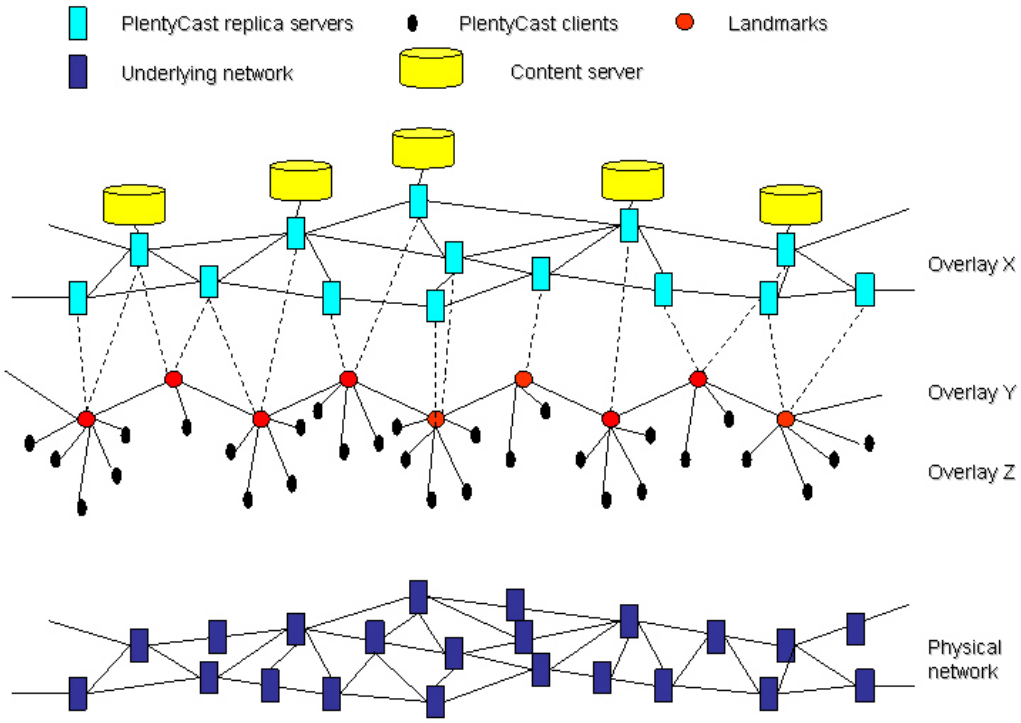


Figure 21. PlentyCast overview

As with most of the CDN topology, PlentyCast consists of two types of servers: replica servers and landmarks. They can be physically co-locate in one site or can be placed in different sites. However, they function in different way, as will be explained in later sections. Therefore, we would like to separate them in Figure 21. PlentyCast overview in order to indicate their differences. In this PlentyCast overlay, we classify all the elements in three different sub-layers. One is overlay X, intuitively, this sub-layer only consists of replica servers. Layer Y consists of landmarks on the edges of the designated ASs. The third layer is formed by PlentyCast clients and is called overlayer Z. An obvious attribute of this layer is the number of elements is not predictable. However, we choose them as part of our CDN strategy in certain circumstances. All three layers are mapped on the underlying physical network. The physical connections can be shared or linked separately. On top of the overlay X, content servers are connected to the entire CDN.

There are six key subsystems and two assistance systems in our architecture. The six key systems includes:

1. PlentyCast client
2. Landmark server
3. Replica server
4. Location and routing system
5. Distribution system
6. Accounting system

6.2 High level system architecture

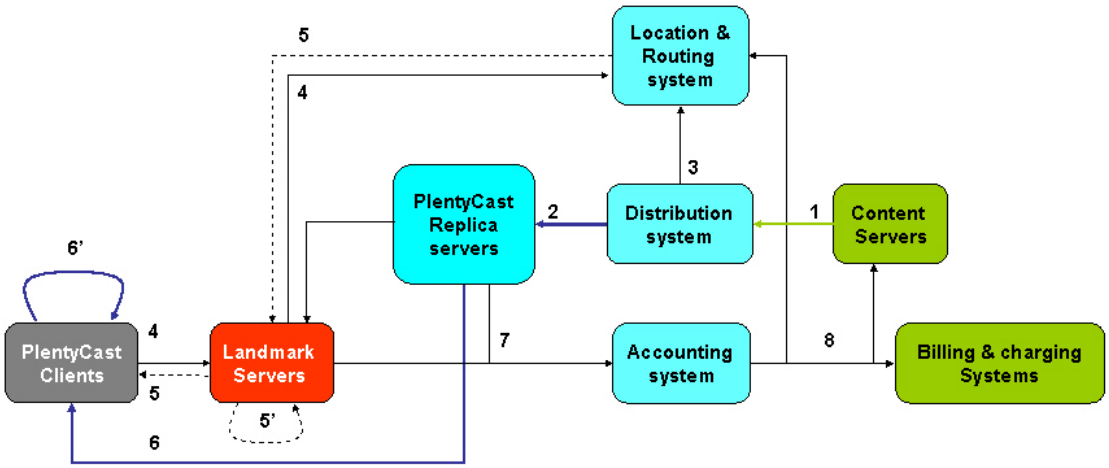


Figure 22. PlentyCast high level architecture

There are another three systems that appear to be external systems of this CDN system. They are content server, billing and charging system. Following the general work flow, the description of this architecture is as follows.

1. Content object copies are sent to the distribution system. In Object Splitter, it is split into fixed length block, and the blocks are coded by the FEC encoder with redundancy information.
2. The distributor places the coded blocks over the replica servers following certain policy.
3. After the distribution has been finished, a notification must be sent to the location and routing system to update the replica locations.
4. When a PlentyCast client requests specific content, the local landmark will forward this request to the location and routing system to select the most suitable replica servers for the client.
5. If the location and routing system find the popularity of the content isn't too high or load of the local replica servers are not high, then a list of local servers will be sent to the PlentyCast client for its selection via the landmark.

   5'. If the location and routing system finds that the popularity of the content is not too high or the load on local servers are very high, then it will instruct the distributor to disseminate more blocks to more replica servers, and then a list of replica servers will be sent to the landmark for reference. Meanwhile, the location and routing system will send an order of swarming delivery to the landmark; the landmark will search amongst its peering group and its local database to lookup blocks existing in any active PlentyCast clients within certain radius of the content space (such as on the same LAN or within the same AS). Then a list of all candidates (servers and PlentyCast peering clients) are sent to the PlentyCast client.

6. According to the selection process in the PlentyCast client, only some replica servers will be selected . Then a many-to-one concurrent download is conducted. The client downloads all the blocks, decodes them, and then reconstructs the content locally.

   6'. According to selection process in the PlentyCast client, some servers and PlentyCast clients will be chosen to start/join swarming delivery. Then a mesh is formed to download and upload the blocks of this content as described in section 3.2.2. The landmark periodically update the status of the clients in the mesh. If the number of blocks of the content exists amongst the PlentyCast clients in the mesh, the landmark shall send a new list of PlentyCast peers the client for re-selection of the nearest blocks.

7. The landmark checks the downloading status periodically of the downloading clients, and then reports to the accounting system about the popularity of this content and the numbers of replicas actual alive in PlentyCast clients. Meanwhile, the replica server also sends the hit rate of the content and its current load to accounting system.

8. The accounting systems manipulate data for location & routing system, billing and charging system, the customer – content provider.

6.3 PlentyCast client

In this user agent, most functions are the same as a web browser. It can for example send an HTTP request for any content which the user wants to access. The difference appears in the following aspects:

6.3.1 Active binning

Every PlentyCast client periodically pings a set of landmarks to obtain RTT between each of the landmarks in their joining phase. This is to used to classify different replica servers into those with the same level; as described in section 2.3.2.1. The level vector for each client is stored and send to the landmark. This information will assist in replica server election.

6.3.2 SNMP client.

To be able to monitor the downloading and/or uploading status in this client, an SNMP MIB[27] for throughput of IP/TCP connections shall be queried.

Mobile agent client

Since the landmark will periodically request the download and/or upload status of the client, a report-on-demand mechanism must exist between the client and the landmark. In study [155], they developed a very efficient way to monitor large scale and dynamic network with small bandwidth consumption. This can be adopted in report-on-demand mechanism between PlentyCast clients and landmarks.

6.3.3 Peer lookup service

If swarming delivery is needed, a PlentyCast client should be able to locate the blocks being downloaded either by one of DHT systems such as Tapestry (as described in section 2.3.1.3) or a platform such as JXTA [117]. In this case, the landmarks is the "super peer". It monitors the downloading and uploading status of each client who is

[27] Management Information Base.

accessing content. It provides a list of PlentyCast clients who are downloading and uploading content when swarming delivery is needed. Thus a PlentyCast client can locate the blocks of that content amongst the peers given by the landmark. In a PlentyCast client, either of these two communication mechanisms shall be implemented.

6.3.4 Peer Selection

Since $n+m$ blocks of content are encoded in FEC codes, these m blocks are redundant blocks. Only n blocks are needed to reconstruct the content. Thereby, selecting n distinct blocks amongst $n+m$ blocks is key in this selection process. In the selection algorithm, the following steps are mandatory in this algorithm:

1. Identify how many blocks are needed to reconstruct this content.
2. Identify the peers and/or servers who carry *any* blocks of the content file, and the number of those peers in the list given by the landmark.
3. Identify the peers who has the download request for this content. Any peer whose portion of the content is lower than 100% shall be uploaded.
4. Selecting the n' peers and/or servers ($n' \leq n$) from the group in 2. The metrics are the peers or servers with:
    a. Shortest distance
    b. Most blocks
    c. Largest bandwidth
    d. Latest download or upload time

    If the total number of blocks accumulated to n or the end of list is reached, then the selection process is ended. Naturally, the rest of the n-n' peer/s in the list are identified as backup peers.
5. Set up connections to those servers and peers in groups of 3 and 4
6. Reading in the blocks from all the hosts in 4 and save, then send out the blocks to all peers in group 3. (It is possible for two peers download and upload happening at once between each other)
7. If the any sending peer/s of the blocks in 1 is disconnected, then promote the redundant peer/s in group 5 to group 4.
8. If the blocks can not be found in any backup peers or there is no back up peers to be selected, then *Peer lookup service* is activated.

6.4 Landmark server

Redirection executor

When a request is received, it should be forwarded to the location and routing system, while a notification is sent to the accounting system to record the object's popularity. If there is a routing request is received from the location and routing system for the PlentyCast client request, the executor shall forward this decision to the client. The decision is usually a list of replica servers who carry the blocks for the client's request. When swarming delivery should be performed, the executor adds the PlentyCast client peers to the list of the replica servers, and then sends the request client/s. A hybrid solution can be chosen amongst the techniques in section 1.4.4.4.2 for the executor.

6.4.1 Placement

The location of the landmarks must be placed at the edges of the ASs. The hierarchy shall be maintained in order to closely map the overlay network to Internet structure.

Thus they can provide good measurement results in *binning* techniques to locate the nearest servers[28] as explained in section 2.3.2.1.

6.4.2 Download & upload monitor

It is very important that the landmark knows the download and upload status of each PlentyCast client in the mesh during the swarming delivery. Thereby, the monitor keeps tracking each PlentyCast peers behavior and their performance. In addition, the degree of content saturation in the mesh must be monitored. This will provide information for peer selecting in the case of node joining and re-selecting on fault happening or on self-organization in the middle of transmission. This can be implemented by either using mobile agent [155] or a platform like JXTA.

6.4.3 Load balancing

If the landmark is overloaded, load balancing shall be devised for the landmarks. A policy of load balancing must be established in the landmark server farm at each site. The following key metrics should be used in such a load balancing process:

- CPU load and memory must be monitored,
- Number of processes in the node must be monitored, and
- Number of PlentyCast clients.

If any of the value of these metrics exceed thresholds which was configured, new request must be forwarded to the neighbor peers of landmarks. It means that landmark shall run on top of a P2P service. I recommend that the same Peer-to-Peer service platform or approaches shall be adopted as the rest of the subsystems such as Peer lookup service and download & upload monitor.

6.5 Distribution system

6.5.1 Object splitter

If there is a copy of a large file is obtained from the content server, it shall be split into fixed length of blocks. Every block shall be in the size range of 32 to 64Kbyte. The design and implementation can be see in the Appendix 1.

6.5.2 FEC encoder

This component is devised to encode the blocks generated by Object Splitter. The rationale is described in section 3.3. The actual implementation could follow the reference model in [156] [157] .

6.5.3 Block distributor

Every block of this content shall be assigned unique ID. Assume l is the number of replica servers, in the first round distribution, if $n+m > l$ then every replica servers gets $(n+m)/l$ copies. If $n+m \leq l$ then randomly choose $l - (n+m)$ blocks from the total $n+m$ blocks and then distributed to the rest of replica servers and make sure there is no duplication during the distribution from each server. This can be done by either Tapestry or JXTA platform.

In the second round distribution, number of the blocks shall be increased in certain replica servers. However, this time the distribution won't be so simple. Since the location and routing system will notice that the popularity of certain content according

[28]How close the severs are depends on how the level vectors have been defined, see section 2.3.2.1

to the report from the replica server and the landmarks in advance. The scale of the dissemination shall be smaller than the first round concerning the replica server load and overlay traffic load, but the portion of the content in this group should be increased.

The location and routing systems shall input a list of the *k* replica servers (in terms of its server selection process) and the portion of the content (*x%*). The same rule as in the first round, variety should be maintained. Thereby the following steps are mandatory in this match-making algorithm.

1. Identify how many blocks are included in this content.
2. Check the server list with the accounting system to find out which block IDs already exist in the servers, and then exclude these blocks.
3. Select the rest of the blocks of the content and put them in a list.
4. Work out the number of the blocks *y* to be added, by the following formula: $y=x\%\cdot(n+m)$.
5. Select the *y* blocks from the list in 3, and assign them to *k* buckets evenly.
6. The *k* buckets shall be sent to the *k* replica servers in the list[29].

6.6 Replica server

6.6.1 Placement

Replica servers shall be placed at the edges of the AS. Preferably, they can be place in the Internet Data Center of ISPs. Since we disseminate variety of blocks of the content across the different servers, the replica server only appears to be a storage for different blocks. In some cases, they even do not get any complete copy of this content. Thus we argue not to choose the approaches in section 1.4.4.1 in order to avoid the NP-hard problem.

6.6.2 Storage and delivery

On one hand, the replica server should store the blocks of the content which are sent by distributor of the distribution system. On the other hand, they shall upload the blocks to the PlentyCast clients or peers who request the content. Whenever it downloads blocks based on the client request, it should cache blocks that have been delivered, in other words, it will increase the numbers of blocks of content in its storage till the entire content is cached locally. It should report the server load, available storage, and the hit rate of specific content to the accounting system.

6.6.3 Cone loading

If there is a list of *k* buckets, they will be assigned to a group of *k* replica servers. Each i^{th} replica server ($i=1,2,3,4,\ldots,k$) should use a Peer-to-Peer look up service to find all the blocks in the i^{th} bucket and download them from peers across the replica overlay. When the i^{th} block is downloading from the peers, meanwhile it should upload the blocks plus the blocks stored on the request client. This procedure, it is named "cone loading". The following Figure depicts this behavior in the overlay.

[29] Another scenario is to let distributor directly establish one-to-many connections and spread the *y* blocks evenly to the servers in a round robin manner. The efficiency of these two scenarios should be compared in future work.

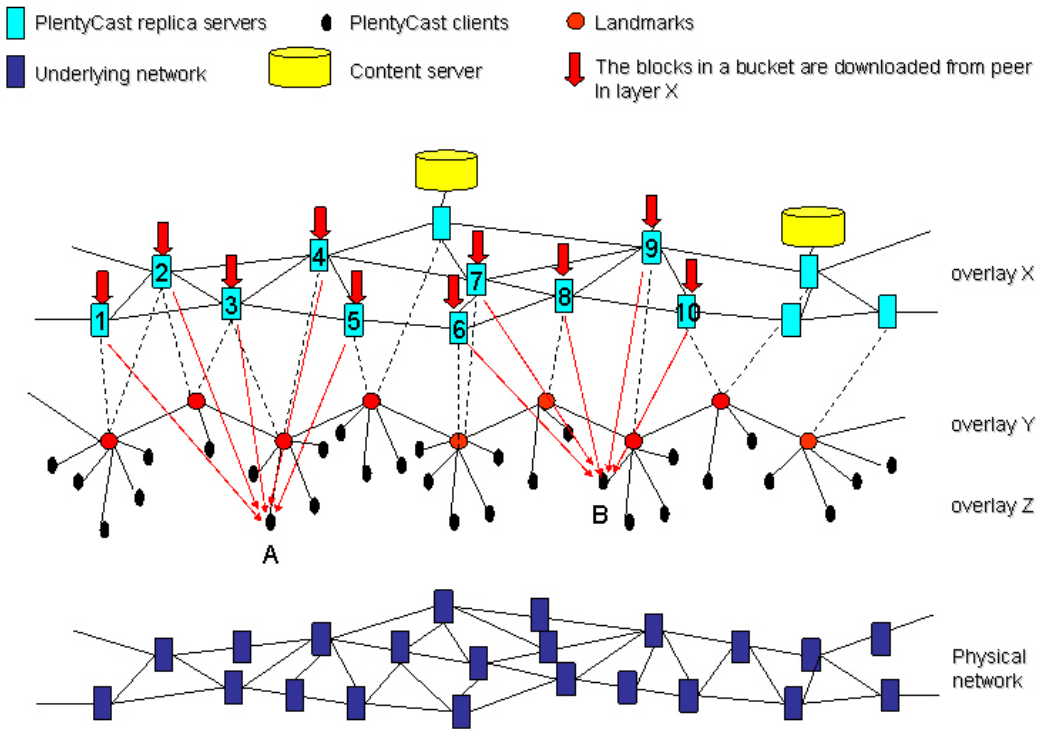


Figure 23. Cone loading

In the above figure, A and B are the request clients for the content. They are concurrently download from the replica servers. The servers are downloading the blocks which they don't have from peers other via Peer-to-Peer lookup service. When replica servers 1-10 are uploading the content blocks, they should cache the blocks which they are downloading from other peers or each other. The traffic generated can be high in the PlentyCast overlay the first time when the content being downloaded. But as more clients request the content, more blocks will be found in the overlay. The bound on the total storage should be considered in the replica server overlay X.

6.6.4 Active binning

Every replica servers should periodically ping a set of landmarks to obtain the RTT between each of the landmarks. This is to used to bin different replica servers who have the same level as described in section 2.3.2.1. The level vectors can be either stored in each replica server or in the database of location and routing systems. This can be adjusted in the future work on simulation.

To achieve low cost and fast content location, we recommend either choosing Tapestry or JXTA as the basic P2P service i.e. to provide peer discovery and content location.

6.7 Location and routing system

6.7.1 Policy engine

The major feature of the location and routing system is to locate suitable replica servers for the user and push content reactively or proactively to the right group/s of replica servers in order to achieve high data availability for the requested content. The following metrics shall be considered in this policy engine: content hit rate and bin.

The small difference between our binning strategy and the one described in section 2.3.2.1 is to add server load and bandwidth to the level vectors. Similarly, we can bin the replica server load and its bandwidth into different levels too. A level vector will be described in the following format:

Since more metrics have been added to the binning, the location and routing system should be able to make a more optimal selection.

If a PlentyCast client request is received, but popularity of the content (i.e., hit rate) exceeds the threshold of PlentyCast system or the load of replica servers appears to be high, an order shall be issued to the landmark so that swarming delivery can be initiated to serve this request.

6.7.2 Server selector

If a PlentyCast client request is received, the server selector will execute the same distributed binning algorithm as described in section 2.3.2.2. There are two additional metrics: server load and bandwidth will be checked by the binning algorithm. The lower the load and the higher their available bandwidth level is, the better servers are for the client. If the servers are selected, a server list will be sent to the client for block selection as described in section 6.3.

6.7.3 Block meter

There should be 100% of the content loaded to this group after the group of servers have been chosen for the client. As described in section 6.6, the replica servers shall be responsible for collecting all the blocks of this content and then uploading them to the requesting client concurrently. The client shall choose the blocks and replicas by itself. This selection shall be applied to the following cases of client requests.

- The first is when the client appears should be a new client (on no blocks of content are downloaded)
- The second occurs when client has already yet partially downloaded the content (portions of blocks) and requests more blocks delivered from replica servers.

6.7.4 Content manager

Periodically, the hit rate of all the content should be checked. If the hit rate of certain content is at a relatively low value, then the number of replica blocks disseminated in the replica servers can be reduced. The following steps are mandatory in such process:

1. Check the hit rate for each specific content object in the accounting system.
2. Sort the content hit rate by different replica servers.
3. Use Peer-to-Peer service to locate the content replicas.
4. Delete the replica for a specific time period (this parameter can be tuned in according to the bound on storage in each server)
5. Update the accounting database.

6.8 Accounting system

There are two classes of data that should be accounted for: hit rate for each element content, and the server load of each replica server. The database header should cover the following mandatory keys as depicted in the following Table 5.

Server ID	Current load	Current bandwidth	Storage capacity	Content ID

Content ID	Current Hit rate	Number of blocks of the content

Table 5. Accountig Database header

The primary key is the Server ID and the secondly key is content ID. In this way, a content ID is bind to the server ID.

6.9 System characteristics analysis and discussion

In this section, we would like to conduct a set of case studies by going through our system with different traffic demands from clients, while we will demonstrate how we can achieve our five goals while processing these different traffic demand in our CDN system.

6.9.1 Case study 1: Normal access mode

In this case, we assume that the content has been published on a web site. The users' access rate does not exceed the bound of the normal access mode. A PlentyCast client in the Internet finds the landmarks in the cache via a probe.

After selecting the three closest landmarks, it records its own bin. Then sends a request (content URL and bin level) to nearest landmark for this content via HTTP. The landmark forwards this request to the location and routing system.

The bin of this client will be compared with the bin of servers in the accounting database. If a similar bin exist, then the servers should be selected for the request. Since this is the first time a client is accessing the content, hit rate won't be considered.

After the servers are chosen, the distribution system must issue a list of buckets for the selected servers. Each replica server locates the their blocks which are contained in the bucket via the P2P lookup service, then downloads them from a replica server peer in overlay X. All new blocks for a replica server will be cached. This list of servers should be sent to the PlentyCast client for the blocks selection by the client. Since this is the first time to download the content, all servers will be chosen to upload the blocks of this content. A *cone loading* will be formed between replica servers grouped with this client. Since the blocks are relative small (32-64 Kbytes per block), downloading and uploading happen between replica server peers, and the replica overlay is relatively stable, the latency until the client gets the complete content should not be longer than a one-to-one download from content server. Another important aspect is that the client will only download n blocks due to the FEC coded blocks, even though the number of blocks prepared in the group of designated servers is $n+m$. Thereby, redundancy plus concurrent download – cone

loading provides both high data availability for content data and reduces user access latency.

Although, this might increase the bandwidth consumption and increase the load on the replicas severs. However, we argue that it only happens when this content is first being accessed. When there are additional clients access this content, more blocks will be distributed to the client from nearby. We estimate that latency would be reduced by several orders of magnitude, and the bandwidth consumption will be reduced due to less and less download happening in the overlay. The only increasing cost is for storage in replica servers as more and more replica of the entire content are created. Fortunately, storage is cheaper than other resources in the network (i.e., rental of a 2M link a year). In addition, our content manager will act as garbage collector which periodically cleans out the deprecated content replicas. Thus, before any flash crowd or DDoS happens, the system increase the number of replicas within the performance bound of server overlay and thus improves content availability and reduces user access latency.

Since we adopt a binning strategy for server selection, this enables clients to choose the server(s) closest to them. This location aware techniques ensure the traffic won't flow over a long distance. It significantly reduces bandwidth consumption on two of the major Internet bottlenecks, specifically, the peering bottleneck and backbone bottleneck on the overlay network. Another advantage of binning is to localize access traffic, thereby it potentially contribute to improve infrastructure performance.

6.9.2 Case study 2: Flash Crowd and DDoS mode

We now examine the system behavior for a high access traffic pattern. In this case, we assume that there are large number of PlentyCast clients accessing specific content via landmarks either in a short period of time or simultaneously (as in DDoS).

The alarm caused by such events are reported to the location and routing system by the accounting system (this is actually reported by the landmarks since the accounting system collected the hit rate values). The location and routing system start the server selection process for the number of routing requests of some of the clients. Based on the number of requests which can **not** be handled, it will issue orders to the landmarks (as if the request were bounced back).

When a landmark receives this order, it will start a swarming delivery process. A mesh for this content will be formed. The binning process in each client gradually causes the access traffic to scale on each LAN. The upload and download policy in each client creates more and more replicas in the mesh. Thus the P2P overlay Z enables high access rate traffic of the clients to be counteracted by themselves (i.e., these clients in turn act as servers to propagate the content).

In the high access rate traffic pattern, the PlentyCast system use P2P swarming delivery techniques to solve the problems of Flash crowds and DDoS. As in case 1, PlentyCast can significantly decrease the user's access latency and reduce bandwidth consumption on the Internet by enabling client access of the content in the same LAN to increase gradually. Given the nature of a hybrid P2P overlay, network scalability has been significantly improved. Since the data torrent has been organized into overlays Z to X, the bottlenecks on peering, backbone and first mile are completely

alleviated. In the last mile, bandwidth consumption can be very high, but the torrent of traffic eventually exist on a LAN (where link layer multicast can be used). Thus it will be easier for a local ISP to handle these events. In a flash crowd, ISP shall does nothing, as the ISP can easily apply the DDoS detection tool such as [159].

6.9.3 Case study 3: ADSL users traffic

In the normal access mode described in section 6.9.1, the PlentyCast client behind an ADSL modern will take the advantage of the broadband downlink – usually with 1-8Mbps bandwidth on the downlink. This and FEC combine to enable very fast download of content.

However, the asymmetry will cause problem as described in section 3.2 when swarming delivery happens. The upstream connection cannot upload the same amount of content as it is downloading. Even worse, because of the details of TCP's rate control, if the upstream path is congested, the downstream performance suffers as well. In PlentyCast, we adopted the mechanisms from both in Swarmcast and BitTorrent.

The block selection process in a PlentyCast client uses distance metrics to choose the number of closest peers for download and upload, this exploits ADSL peers located in the same LAN. Thus the TCP performance has been improved and we can potentially avoid upstream being clogged. In addition, since the block size has been limited to a quite small size (32-64Kbytes per block), this also alleviates the last-mile bottleneck.

We exploit the chock algorithm from BitTorrent [4]. As an exception to simultaneous upload and download, a PlentyCast can stop uploading to specific peers. If a PlentyCast client is located behind the ADSL modern, it will execute *Pause* algorithm for swarming delivery. Every 10 seconds, the ADSL peer will evaluate the upload rate for the peers who upload to it. It will choose the of peers whose sum of upload rate equals its uplink rate. Then it will download only from these peers, but stop others. To save overhead during each TCP session, the ADSL peer open the connection to those paused-download peers. Every 30 seconds, the ADSL peer will resume one of the paused-download peer regardless that peer's upload rate. In this way, the ADSL peer alleviates the bottleneck brought by TCP "fairness" attribute.

In accordance with the result in study [11], when different users (dialup, broadband, and office users) co-exist in the same data mesh, the performance of swarming performance in each class of users won't be effected significantly. In PlentyCast, we actually periodically makes each ADSL peer converge just as a symmetric peer whose download and upload rate trend to be equal mean while we will not degrade the advantage of broadband i.e., the download link can be saturated during another period of time. Thus our system can contribute to enhanced performance of the infrastructure.

6.9.4 System characteristics

Since PlentyCast is a new type of CDN and primarily uses techniques of P2P, Agent, and Error correction coding, we argue that the following criteria and attributes in CDN and P2P (described in section 1.2.3 and section 2.2) should be used to evaluate our system. They are :

1. Fast access

2. Robustness
3. Transparency
4. Efficiency
5. Adaptive
6. Reliability
7. Simplicity
8. Security
9. Decentralization
10. Scalability
11. Self-organization
12. Anonymity
13. Cost of ownership
14. Ad-hoc connectivity
15. Performance
16. Digital Rights Management
17. Usability and Transparency
18. Fault-resilience
19. Manageability
20. Interoperability

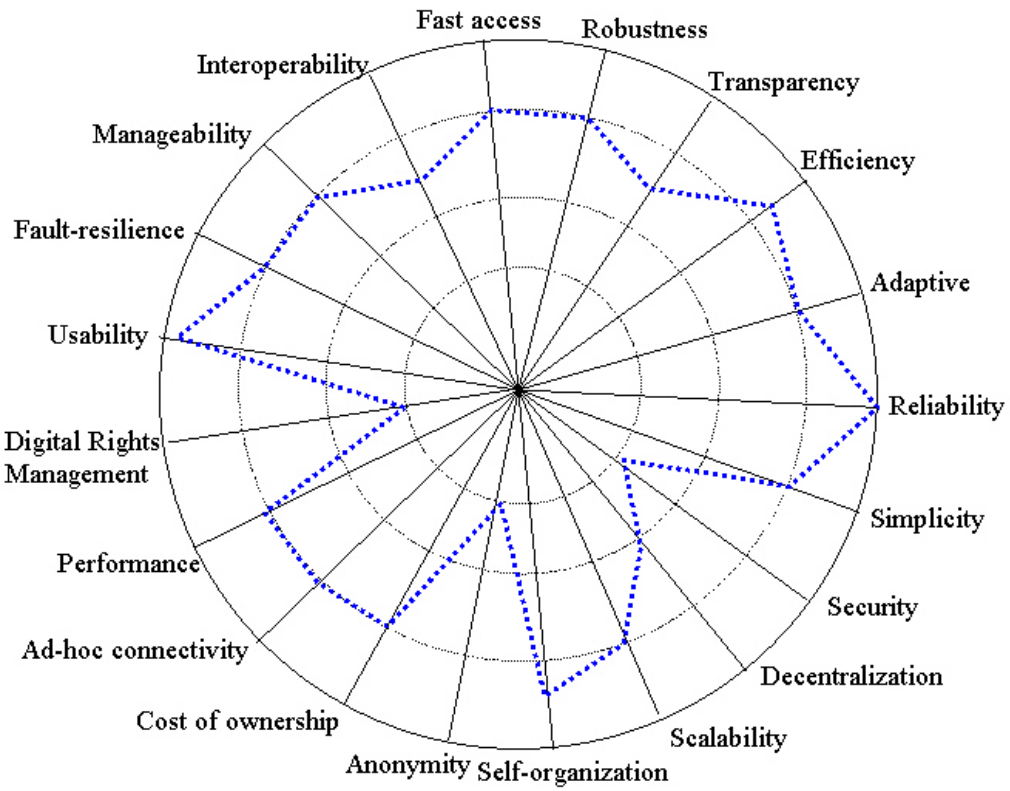


Figure 24. PlentyCast system characteristics

Using the above criteria, we can define 5 grades for each of these attributes. From the highest to the lowest: Very good, good, acceptable, not good, and worse. Table 5 will explain our motivation to state our systems characteristics:

	Very good	Good	Acceptable	Not good	Worse
Fast access		▪ Cone loading ▪ Swarming download ▪ intelligent location& routing			
Robustness		▪ High content availability ▪ Fault resilient			
Transparency			▪ PlentyCast inherited from web browser ▪ Small configuration problems in NAT and firewalls		
Efficiency	▪ FEC reduces overheads ▪ Good solution to Internet bottlenecks				
Adaptive		▪ Flash crowd and DoS aware ▪ Intelligent location and routing ▪ Extended Binning strategy			
Reliability	▪ Very high content availability ▪ Load balancing design in landmark ▪ Intelligent location and routing				
Simplicity		Most of techniques are verified by good practices			
Security				▪ Security in the signaling system ▪ Enforced data integrity when using secured hashing	
Decentralization			This is a hybrid architecture design		
Scalability		P2P service design			
Self-organization		▪ Intelligent location and routing system			

		▪ Automatic switch between normal mode and high traffic mode ▪ Localize the intensive traffic in flash crowd and DoS ▪ Pause algorithm for ADSL users			
Anonymity				Not yet designed	
Cost of ownership		▪ Intelligent location and routing system ▪ Mechanism for cost sharing in Flash crowd and DoS ▪ Swarming delivery ▪ Landmark load sharing			
Ad-hoc connectivity		▪ Swarming delivery with download and upload strategy ▪ Aggressive content / peer selection algorithm			
Performance		▪ Adaptive distribution system ▪ Replica server caching ▪ Intelligent routing			
DRM				Not yet designed	
Usability	▪ Low access latency ▪ High content availability ▪ Good network adaptive				
Fault-resilience		▪ High content availability by using FEC codes ▪ Concurrent download ▪ Intelligent location and routing			

Manageability		▪ Accounting system for whole system ▪ Landmarks for self-management in swarming			
Interoperability			▪ Pause algorithm to deal with ADSL in swarming delivery ▪ Need to developed solution for NAT and firewall		

Table 6. System characteristics clarification

Chapter 7 Conclusion and future work

In this project, we have made an intensive literature study in the area of CDN, P2P, Swarming, and Error Correction coding. Based on our findings, we proposed a novel CDN architecture to achieve: improved access latency, improve network scalability, improve content availability, lower bandwidth consumption, improve infrastructure performance. Finally, we find that on our CDN – PlentyCast can in theory accomplish all the goals.

7.1 Conclusion

Large content distribution has become a key issue in the Internet. P2P file-sharing techniques have been recognized as very efficient for large content distribution. Swarming content delivery increases user access speed by several orders of magnitude. It is inevitable that P2P swarming techniques will be used in the next generation CDNs. One signal is very clear, a new generation of CDN will emerge very soon. Big companies like Intel, have already built P2P services to use **all hosts** to share large content across the enterprise network [160]. In media industry, large content streaming delivery over Internet become an inevitable trend. Meanwhile, the war between traditional distribution & record industry and P2P companies & users becomes a barrier to P2P distribution of content on the Internet (for instance, consider the battle between R1AA and P2P companies in US.A.). However, P2P delivery will soon be very significant because of its attractive features for the Internet users – which is to deliver large content quickly.

7.2 Future work

First of all, we will build a software prototype system in the very near future. And this prototype will be tested for a real event soon in 2005. An intensive evaluation test will be conducted for all the features we described in the PlentyCast architecture.

There are still open questions in the area of large content distribution such as:

(i) How to deliver dynamic large content, for example real time streaming media distribution. The biggest challenge is real time streaming content distribution is the synchronization between voice stream and video steam over a long distance in terrestrial networks. The longer the distance is, the harder they can be synchronized. In addition, how to improve the user access rate for such content becomes even harder. To our best knowledge, there is no optimal solution in this area. The only known attempts is the MDC (Multiple Description Coding) solution in Microsoft CoopNet [8].

(ii) How to efficiently protect these applications against malicious peers. DDoS attack is firstly using email to propagate the virus to large number of innocent clients, then activate the web request on certain day to a targeted web server. In our architecture, we use landmarks to manage the overlay Z. If there is malicious PlentyCast clients having DDoS to targeted landmarks, this will have very serious consequences. We must investigate how to handle this kind of situation.

(iii) What AAA mechanism shall be used in such CDN for limited distribution. PlentyCast client shall be authenticated and authorized and accounted in an efficient way in order to avoid problem in (ii) and provide base for comprehensive DRM solutions for some specific content distribution

(iv) How to develop Digital Rights Management solution based on such CDN. Most of content can be distributed without any copy right or license concerns but some specific content such as licensed software and movie file with copy righted etc. If the contents shall not be infringed and only available for viewing then a DRM solution must be developed in our CDN.

(v) How to run a good business model for this type of CDN. What we addressed in business model is about the value chain amongst users, Content providers, and ISPs. Firstly, if the users use our network, they contribute their resources (CPU, memory, and storage) for delivery. Actually they have already paid something whenever they download. So this avoids problem that "need users to pay" which most users dislike. Secondly, as a content provider, he or she is still paying to the CDN operator for content distribution, but they can pay _less_ than today because today's CDN is considered as an expensive solution, not all web content providers can use. Thirdly, the CDN operators shall pay _less_ than today either because their cost has been shared with the users and our solution save bandwidth in their network. Fourthly, the tier 1 and 2 ISPs' traffic engineers are happy to have relatively predictable traffic pattern appears in their dimension tools. The Tier-3 local ISP in our solution will have more traffic on their LANs. But this is much better than before. Today's Tier-3 ISP face bother User's large traffic and expensive long haul links to their Tier-2 or 3 ISPs. Our solution actually alleviated downlink bottleneck. The LAN network expansion shall be more fun than their long haul expansion. For example, they may ask their subscriber to expand their access network to higher capacity links such as FDDI, wireless LAN etc. However, a deep investigation on this is needed.

Introduction to the author:

Cao Weiqiu (Mike) graduated with his computer application B. Eng. degree in China National University of Defense Technology in July 1994. He used to work in Ericsson from 1994 to 2002. During the course, he worked with switches, optical transmission systems, and network management systems of Telecommunication as both engineer and managerial roles. When studying in Royal Institute of Technology (KTH), he had examined internetworking technology in many aspects.

Reference

[1] Jia Wang, A survey of Web Caching Schemes for the Internet. ACM SIGCOMM Computer Communication Review. Volume 29 , Issue 5 (October 1999) Pages: 36 - 46

[2] Gang Peng, CDN: Content Distribution Network http://citeseer.ist.psu.edu/peng03cdn.html accessed on 2004-03-12 18:25

[3] www.akamai.com accessed on 2004-03-12 18:22

[4] Bram Cohen, Incentives Build Robustness in BitTorrent http://www.google.com/url?sa=U&start=1&q=http://bitconjurer.org/BitTorrent/bittorrentecon.pdf&e=1102 accessed on 2004-03-12 18:20

[5] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja1, Jim Pruyne, Bruno Richard, Sami Rollins 2 , Zhichen Xu. Peer-to-Peer Computing. HP Laboratories Palo Alto. HPL-2002-57 (R.1) July 3rd , 2003

[6] Kunwadee Sripanidkulchai, Bruce Maggs, and Hui Zhang. Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems, Infocom 2003.

[7] Elisa Turrini, Fabio Panzieri. Using P2P Techniques for Content Distribution Internetworking: A Research Proposal http://csdl.computer.org/comp/proceedings/p2p/2002/1810/00/18100171.pdf accessed on 2004-03-12 18:17

[8] http://research.microsoft.com/~padmanab/projects/coopnet/ accessed 2004-01-05 18:01

[9] http://www.esi.org/ accessed in 2004-01-05 19:08

[10] Venkata N. Padmanabhan, Helen J. Wang, and Philip A. Chou. Distributing Streaming Media Content Using Cooperative Networking. Accessed in http://research.microsoft.com/projects/coopnet/

[11] Daniel Stutzbach, Daniel Zappala, and Reza Rejaie, Swarming: Scalable Content Delivery for the Masses. January, 2004 (Technical Report, UO-CIS-TR-2004-1).

[12] Akamai white paper – Internet Bottlenecks Accessed on 2004-01-6 22:22 http://www.akamai.com/en/html/services/white_paper_library.html

[13] Broadband Access in Korea: Experience and Future Perspective by Yong-Kyung Lee and Dongmyun Kee, IEEE Communications Magazine, December 2003, pp. 30-36.

[14] Controlling P2P Traffic, accessed on LIGHT Reading on 2004-01-07 01:05 http://www.lightreading.com/document.asp?doc_id=44435

[15] www.p-cube.com. Accessed on 2004-01-07 01:16am

[16] http://www.calresco.org/sos/sosfaq.htm#1.2 accessed on 2004-01-07 01:38am

[17] Analyzing Peer-to-Peer Traffic Across Large Networks, by Shubho Sen and Jia Wang, to appear in *ACM/IEEE Transactions on Networking*, 2004. Accessed on 2004-01-07 20:27

[18] http://www.openp2p.com/pub/q/p2p_category Access on 2004-01-07 20:45

[19] http://www.eweek.com/article2/0,4149,1431490,00.asp accessed on 2003-01-11 11:37

[20] Craig Labovitz, G. Robert Malan, and Farnam Jahanian, Originals of Internet Instability, http://citeseer.nj.nec.com/labovitz99origins.html , accessed on 2004-01-16 17:00

[21] http://www.ietf.org/rfc/rfc894.txt and http://www.ietf.org/rfc/rfc1042.txt, accessed on 2004-01-16 18:25

[22] Katia Obraczka and Fabio Silva, Network Latency Metrics for Server Proximity http://citeseer.nj.nec.com/obraczka00network.html, accessed on 2004-01-16 19:34
[23] Mohammed J., Kabir, Apache Server Administrator's Handbook, IDG Books, pp.68-69,
[24] Charles D. Cranor et al., Enhanced Streaming Services in a Content Distribution Network, IEEE Internet Computing, July 2001.
[25] M. Charikar, and S. Guha. Improved Combinatorial Algorithms for the Facility Location and K-Median Problems. In *Proc. of the 40th Annual IEEE Conference on Foundations of Computer Science*, 1999.
[26] Lili Qiu, Venkata N. Padmanabhan, and Deoffery M. Voelker. On the placement of web server replicas. In Proceedings of IEEE INFOCOM 2001 Conference, Anchorage, Alaska USA, April 2001.
[27] Year Brutal. Probabilistic approximation of metric space and its algorithmic applications. In 37th Annual IEEE Symposium on Foundations of Computer Science, October 1996
[28] Vijay Varian. Approximation methods. Springer-Verlag, 1999
[29] Pavlin Radoslavov, Ramesh Govindan, and Deborah Estrin Topology-Informed Internet Replica Placement (2001) Proceedings of WCW'01: Web Caching and Content Distribution Workshop, Boston, MA
[30] K. Kangasharju, J. Roberts, and K. Ross. Object replication strategies in content distribution networks. In 6th International Web Caching Workshop and Content Delivery Workshop, Boston, MA, 2001.
[31] Gary Audin. Reality check on five-nines. *Business Communications Review*, pages 22–27, May 2002.
[32] Madhukar R. Korupolu Michael Dahlin Coordinated Placement and Replacement for Large-Scale Distributed Caches (1998)
[33] Jussi Kangasharju, James Roberts, and Keith W. Ross, Object Replication Strategies in Content Distribution Networks (2001)
[34] http://mathworld.wolfram.com/NP-HardProblem.html accessed on 2004-02-01 20:55
[35] http://www.nist.gov/dads/HTML/zipfslaw.html accessed on 2004-02-01 21:17
[36] Yan Chen, Lili Qiu, Weiyu Chen, Luan Nguyen, Randy H. Katz, Efficient and Adaptive Web Replication using Content Clustering. http://citeseer.nj.nec.com/596760.html accessed on 2004-02-02 13:03
[37] M. Kafisson, C. Karamanolis, and M. Mahalingam. A Framework for Evaluating Replica Placement Algorithm. Technical report, HP Lab, 2002. http://www.hpl.hp.com/personal/Magnus_Karlsson/papers/rp_framework.pdf accessed on 2004-02-02 14:45
[38] Hyper Text Transfer Protocol HTTP RFC 1945 http://www.ietf.org/rfc/rfc1945.txt?number=1945 accessed on 2004-01-19 22:24
[39] http://en.wikipedia.org/wiki/OSI_model accessed on 2004-02-06 19:57
[40] http://www.gnutella.com/ accessed on 2004-02-07 15:51
[41] http://www.kazaa.com/us/index.htm accessed on 2004-02-07 15:51
[42] http://www.freenetproject.org/ accessed on 2004-02-07 15:51
[43] D. R. Boggs. Internet Broadcasting. PhD thesis, Electrical Engineering Dept., Stanford University, January 1982. Also Tech. Rep. CSL-83-3, Xerox PARC, Palo Alto, Calif.

[44] C. Partridge, T. Mendez, and W. Milliken. RFC 1546: Host Anycasting service, November 1993 http://www.ietf.org/rfc/rfc1546.txt?number=1546 accessed 2004-02-07 18:53
[45] Z. Fei, S. Bhattacharjee, E. Zegura, and M. Ammar. A Novel Server Selection Technique for Improving the Response Time of a Replicated Service. In Proc. of IEEE Infocom, March 1998.Page 783-791
[46] http://www.ietf.org/rfc/rfc3568.txt?number=3568 RFC 3568 Known Content Network (CN) Request-Routing Mechanisms. Accessed on 2004-02-07 21:27
[47] Domain Name Service DNS RFC 1591 http://www.ietf.org/rfc/rfc1591.txt?number=1591 accessed on 2004-01-25 21:26
[48] IAB Architectural and Policy Considerations for Open Pluggable Edge Services http://www.ietf.org/rfc/rfc3238.txt?number=3238 accessed on 2004-02-08 9:59
[49] E. Cohen, B. Krishnamurthy, and J. Rexford, Efficient algorithms for predicting requests to Web servers, Proceedings of Infocom'99.
[50] A. Dingle and T. Partl. Web cache coherence. In Proc Fifth International WWW Conference, May 1996.
[51] RFC 791 http://www.ietf.org/rfc/rfc0791.txt?number=791 accessed on 2004-02-07 16:04
[52] V. Cate, Alex - a global file system, Proceedings of the 1992 USENIX File System Workshop, pp. 1-12, May 1992.
[53] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrel, A hierarchical Internet object cache, Usenix'96, January 1996.
[54] A. Luotonen and K. Altis, World Wide Web proxies, Computer Networks and ISDN Systems, First International Conference on WWW, April 1994.
[55] D. Wessels, Intelligent caching for World-Wide Webobjects, Proceedings of INET'95, Honolulu, Hawaii, June 1995.
[56] B. Krishnamurthy and C. E.Wills, Study of piggyback cache validation for proxy caches in the World Wide Web, Proceedingsof the 1997 USENIX Symposium on Internet Technology and Systems, pp. 1-12, December 1997.
[57] B. Krishnamurthy and C. E. Wills, Piggyback server invalidation for proxy cache coherency, Proceedings of the WWW-7 Conference, pp. 185-194, 1998.
[58] B. Krishnamurthy and C. E. Wills, Proxy cache coherency and replacement - towards a more complete picture, ICDC99, June 1999.
[59] V. Duvvri, P. Shenoy, and R. Tewari. Adaptive leases: A strong consistency mechanism for the world wide web. In INFOCOM, 2000.
[60] Z. Fei. A Novel Approach to Managing Consistency in Content Distribution Networks**.** In Proceedings of the 6th Workshop on Web Caching and Content Distribution, Boston, MA, June 2001.
[61] J. Chuang and M. Sirbu, "Pricing multicast communication: A cost based approach," in Proceedings of INET'98, July 1998. Geneva, Switzerland.
[62] John Dilley, Martin Arlitt, Stephane Perret, and Tai Jin. The Distributed Object Consistency Protocol. Technical report, Hewlett-Packard Labs Technical Reports, 1999.
[63] http://thewhir.com/marketwatch/nas010604.cfm accessed on 2004-01-11 13:54
[64] http://www.eweek.com/print_article/0,3048,a=116052,00.asp accessed on 2004-02-09 19:49

[65] http://www.contentalliance.org/docs/draft-green-cdnp-gen-arch-02.html accessed on 2004-02-09 20:01
[66] http://www.ietf.org/rfc/rfc3570.txt?number=3570 accessed on 2004-02-09 20:04
[67] http://www1.ietf.org/mail-archive/ietf-announce/Current/msg24223.html accessed on 2004-02-09 20:07
[68] Alexandros Biliris, Chuck Cranor, Fred Douglis, Michael Rabinovich, Sandeep Sibal, Oliver Spatscheck, Walter Sturm. CDN Brokering. Proceedings of WCW'01
[69] http://www.etouch.net/ accessed on 2004-02-09 20:20
[70] http://www.speedera.com/ accessed on 2004-02-09 20:22
[71] http://www.sprint.com/ accessed on 2004-02-09 20:23
[72] K. Lee, S. Chari, A. Shaikh, S. Sahu, and P. Cheng. Protecting Content Distribution Networks. IBM Research Report RC 22566, September
[73] http://staff.washington.edu/dittrich/misc/ddos/ accessed 2004-02-09 20:50
[74] L. Amini, A. Shaikh, H. Schulzrinne. Effective Peering for Multi-provider Content Delivery Services. to appear in Proc. of IEEE INFOCOM, June 2004.
[75] http://www.cisco.com/ accessed on 2004-02-10 17:04
[76] http://newsroom.cisco.com/dlls/innovators/content_netwrk/boomerang.html accessed on 2004-02-10 16:54
[77] http://www.cisco.com/univercd/cc/td/doc/pcat/cr4450.htm accessed on 2004-02-10 18:21
[78] http://en.wikipedia.org/wiki/Peer-to-Peer accessed on 2004-02-10 21:36
[79] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja1, Jim Pruyne, Bruno Richard, Sami Rollins 2 , Zhichen Xu. Peer-to-Peer Computing. HP Laboratories Palo Alto. HPL-2002-57 (R.1) July 3rd , 2003
[80] http://en.wikipedia.org/wiki/Pattern accessed 2004-02-11 19:35
[81] http://en.wikipedia.org/wiki/Scalability accessed on 2004-01-11 21:46
[82] Sylvia Ratnasamy, Mark Handley, Richard Karp, Scott Shenker. Application-level Multicast using Content-Addressable Networks (2001). Lecture Notes in Computer Science. Proceedings of the Third International COST264 Workshop on Networked Group Communication table of contents. Pages: 14 - 29
[83] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications (2001).
[84] http://oceanstore.cs.berkeley.edu/ accessed on 2004-02-11 23:18
[85] P. Druschel and A. Rowstron. "Past: Persistent and anonymous storage in a peer-to-peer networking environment," in Proc. the 8th IEEE Work. Hot Topics in Operating Systems (HotOS), Germany, May 2001, pp. 65--70.
[86] Principia Cybernetica Web http://pespmc1.vub.ac.be/SELFORG.html accessed on 2004-02-11 23:24
[87] Ben Y. Zhao, Yitao Duan, Ling Huang Anthony D. Joseph, John D. Kubiatowicz. Brocade: Landmark Routing on Overlay Networks (2002).
[88] ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, distributed object location and routing for large-scale Peer-to-Peer systems. In Proceedings of IFIP/ACM Middleware 2001 (November 2001).
[89] Zhao B. Y., Kubiatowicz J. D., and Joseph A. D. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB/CSD-01-1141, UC Berkeley, EECS, 2001.

[90] http://setiathome.ssl.berkeley.edu/ accessed on 2004-02-12 15:37
[91] http://bitconjurer.org/BitTorrent/ accessed on 2004-02-12 15:39
[92] http://www.napster.com/ accessed on 2004-02-12 15:50
[93] Bryce Wilcox-O'Hearn. Experience deploying a large-scale emergent network First international workshop, IPTPS 2002 Cambridge, MA, USA, March 2002, LNCS 2429
[94] Beverly Yang, Hector Garcia-Molina. Comparing Hybrid Peer-to-Peer Systems (2001). The VLDB Journal
[95] S. Milgram, "The small world problem," Psychology Today, 61(1) (1967).
[96] The Small World Web. 2000. Adamic, L. Technical Report, Xerox Palo Alto Research Center.
[97] M. K. Ramanathan, V. Kalogeraki, J. Pruyne, Finding Good Peers in the Peer-to-Peer Networks. International Parallel and Distributed Computing Symposium, 2001. Fort Lauderdale, Florida, April 2002.
[98] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems (2002). Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)
[99] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance (1999). OSDI: Symposium on Operating Systems Design and Implementation
[100] http://www.ietf.org/rfc/rfc2989.txt?number=2989 accessed on 2004-02-12 21:15
[101] http://en.wikipedia.org/wiki/Digital_rights_management accessed on 2004-02-12 21:16
[102] http://www.ietf.org/rfc/rfc1631.txt?number=1631 accessed on 2004-02-12 21:21
[103] http://www.ietf.org/rfc/rfc3093.txt?number=3093 accessed on 2004-02-12 21:25
[104] G. Coulouris, J. Dollimore, and T. Kindberg. Distributed Systems Concepts and Design. Addison-Wesley, third edition, 2001.
[105] S. Katzenbeisser and F.A.P. Petitcolas. Information Hiding: Techniques for Steganography and Digital Watermarking," Artech House.
[106] http://www.riaa.com/ accessed on 2004-02-13 08:15
[107] Miguel Castro. Practical Byzantine Fault Tolerance (2001). OSDI: Symposium on Operating Systems Design and Implementation
[108] http://www.stanford.edu/group/pandegroup/genome/ accessed on 2004-02-13 9:47
[109] http://www.endeavors.com/securecollaboration.html accessed on 2004-02-13 10:31
[110] http://www.groove.net/ accessed on 2004-02-13 10:33
[111] http://www.yahoo.com/ accessed on 2004-02-13 10:50
[112] http://web.icq.com/ accessed on 2004-02-13 10:52
[113] S. Halabi and D. McPherson. Internet Routing Architectures. Cisco Press, 2nd edition., 2001. Chapter 6 Tuning BGP Capabilities
[114] Ludmila Cherkasova and Jangwon Lee. FastReplica: Efficient Large File Distribution within Content Delivery Networks. HPL-2003-43 20030319 External
[115] Marc Waldman, Aviel Rubin, and Lorrie Cranor. Publius: A robust, tamper-evident, censorship-resistant web publishing system (2000). Proc. 9th USENIX Security Symposium

[116] http://en.wikipedia.org/wiki/Interoperability accessed on 2004-02-13 18:41
[117] http://www.jxta.org/ accessed on 2004-02-13 18:43
[118] http://www.beepcore.org/beepcore/docs/wp-p2p.jsp accessed on accessed on 2004-02-13 18:45
[119] Siu Man Lui and Sai Ho Kwok, Interoperability of Peer-To-Peer File Sharing. ACM SIGecom Exchanges Volume 3 , Issue 3 Summer, 2002. Pages: 25 – 33,
[120] http://www.audiogalaxy.com/ accessed on 2004-02-14 16:15
[121] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications table of contents. Karlsruhe, Germany SESSION: Peer-to-peer table of contents Pages: 381 - 394 2003
[122] C. Greg Plaxton, Rajmohan Rajaraman, and Andréa W. Richa. Accessing nearby copies of replicated objects in a distributed environment. ACM Symposium on Parallel Algorithms and Architectures.
[123] Roberto Rinaldi and Marcel Waldvogel. Routing and Data Location in Overlay Peer-to-Peer Networks. IBM Research. Zurich Research Laboratory
[124] David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Christopher Wells, Ben Zhao, and John Kubiatowicz. OceanStore: An Extremely Wide-Area Storage System (2000). http://citeseer.ist.psu.edu/bindel00oceanstore.html accessed on 2004-03-12 17:46
[125] A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and B.Welch. The Bayou architecture: Support for data sharing among mobile users. In *Proc. of IEEE Workshop on Mobile Computing Systems & Applications*, Dec. 1994.
[126] Sylvia Ratnasamy, Mark Handley, Richard Karp, Scott Shenker. Topologically-Aware Overlay Construction and Server Selection (2002). Proceedings of IEEE INFOCOM'02
[127] Xin Yan Zhang, Qian Zhang, Member, Zhensheng Zhang, Gang Song and Wenwu Zhu. A Construction of Locality-Aware Overlay Network: mOverlay and Its Performance. IEEE IEEE Journal on Selected Areas in Communications.
[128] Rolf Winter, Thomas Zahn, and Jochen Schiller. Institute of Computer Science. Topology-Aware Overlay Construction in Dynamic Networks. Freie Universität Berlin, Germany
[129] Miguel Castro, Peter Druschel, and Y. Charlie Hu. Topology-aware routing in structured peer-to-peer overlay networks (2002). Antony Rowstron. Microsoft Research
[130] S. Hotz, "Routing information organization to support scalable routing with heterogeneous path requirements," Tech. Rep. PhD thesis (draft), University of Southern California, 1994.
[131] http://en.wikipedia.org/wiki/Artificial_intelligence accessed on 2004-02-19 11:57
[132] http://www.molbio.ku.dk/MolBioPages/abk/PersonalPages/Jesper/Swarm.html accessed on 2004-02-19 23:05

[133] http://www.msci.memphis.edu/~franklin/AgentProg.html accessed on 2004-02-21 12:12
[134] http://tracy.informatik.uni-jena.de/research.html accessed on 2004-02-21 12:12
[135] Russell, Stuart J. and Norvig, Peter, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995.
[136] http://en.wikipedia.org/wiki/Flash_crowd accessed on 2004-02-124 20:33
[137] K. Kong, and D. Ghosal. Mitigating server-side congestion in the Internet through Pseudoserving. IEEE/ACM Transactions on Networking 7, 4 (Aug. 1999), 530-544.
[138] Tyron Stading, Petros Maniatis, and Mary Baker. Peer-to-Peer Caching Schemes to Address Flash Crowds. In 1st International Workshop on Peer-to-Peer Systems (IPTPS 2002), March 2002.
[139] Angelos Stavrou, Dan Rubenstein, and Sambit Sahu. A Lightweight, Robust P2P System to Handle Flash Crowds. In *IEEE ICNP*, November 2002.
[140] Fast File Splitter: http://www.maros-tools.com/products/FFS/ accessed on 2004-02-25 10:26
[141] Turbo File Split: http://www.computer-software.org/utilities/file_compression/turbo_file_split.asp accessed on 2004-02-25 10:22
[142] NET energy http://www.computer-software.org/utilities/file_compression/turbo_file_split.asp accessed on 2004-02-25 10:27
[143] Easy File Splitter http://www.filesplitter.net/ accessed on accessed on 2004-02-25 10:27
[144] http://onionnetworks.com/ accessed on 2004-02-27 9:50
[145] http://sourceforge.net/projects/swarmcast/ accessed 2004-02-27 9:51
[146] http://bitconjurer.org/BitTorrent/protocol.html accessed 2004-02-27 11:25
[147] http://www.cs.ucr.edu/~csyiazti/courses/cs204/project/html/final.html#foot1728 accessed on 2004-02-29 23:25
[148] Matei Ripeanu and Ian Foster, Mapping Gnutella Network**,** 1st International Workshop on Peer-to-Peer Systems, Cambridge, Massachusetts, March 2002
[149] I. S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Field", J. SIAM, vol. 8, pp. 300-304, 1960.
[150] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, J. Crowcroft. RFC 3453: The Use of Forward Error Correction (FEC) in Reliable Multicast. December 2002 http://www.ietf.org/rfc/rfc3453.txt?number=3453 accessed on 2004-03-04 18:26
[151] D. Patterson, G. Gibson, R. Katz. A case for redundant arrays of inexpensive disks (RAID), Proceedings of ACM SIGMOD '88, 1988.
[152] Ann Chervenak. Tertiary Storage: An Evaluation of New Applications PhD thesis in University of California at Berkeley (1994) PhD thesis
[153] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Efficient Erasure Correcting Codes", IEEE Transactions on Information Theory, Special Issue: Codes on Graphs and Iterative Algorithms, pp. 569-584, Vol. 47, No. 2, February 2001.

[154] J. Lacan, L. Lancérica, and L. Dairaine, When FEC Speed up Data Access in P2P Networks. Proceedings of IDMS'02 Conference (Interactive Distributed Multimedia Systems), Coimbra, Portugal, 2002
[155] Koon-Seng Lim and Rolf Stadler. Developing pattern-based management programs. CTR Technical Report 503-01-01 August 6, 2001 http://www.ctr.columbia.edu/~stadler/papers/MMNS01-Lim-Stadler-Long.pdf accessed on 2004-03-05 10:49
[156] http://www.fokus.gmd.de/research/cc/mobis/products/fec_old/content.html accessed on 2004-03-07 16:15
[157] http://info.iet.unipi.it/~luigi/fec.html accessed on 2004-03-07 16:16
[158] Patric Hadenius. "Relieving Peer-to-Peer Pressure" Technology Review (02/25/04) http://www.technologyreview.com/ accessed on 2004-02-28 20:00
[159] http://www.securiteam.com/tools/5BP0G000IW.html accessed on 2004-03-10 21:48
[160] http://www.intel.com/business/newstech/peertopeer.pdf accessed on 2004-03-12 15:32
[161] FTP stands for File Transfer Protocol RFC http://www.w3.org/Protocols/rfc959/Overview.html accessed on 2004-02-07 23:09
[162] RTSP stands for Real Time Streaming Protocol http://www.ietf.org/rfc/rfc2326.txt?number=2326 accessed on 2004-02-07 23:09
[163] http://www.ietf.org/rfc/rfc3568.txt?number=3568 accessed 2004-02-07 22:31
[164] RFC 2782 http://www.ietf.org/rfc/rfc2782.txt?number=2782 accessed on 2004-02-07 23:34
[165] RFC 2246 http://www.ietf.org/rfc/rfc2246.txt?number=2246 accessed on 2004-02-07 23:40
[166] http://en.wikipedia.org/wiki/Peer-to-Peer accessed on 2004-04-02 18:31
[167] http://legion.virginia.edu/ accessed on 2004-04-04 20:18
[168] http://www.ietf.org/rfc/rfc2914.txt?number=2914 accessed on 2004 04-05 10:40

Appendix 1: An typical program of how to split a large file into pieces

The `split` program splits large text files into smaller pieces. It is written in `Awk`. By default, the output files are named `` `xaa' ``, `` `xab' ``, and so on. Each file has 1000 lines in it, with the likely exception of the last file. To change the number of lines in each file, you supply a number on the command line preceded with a minus, e.g., `` `-500' `` for files with 500 lines in them instead of 1000. To change the name of the output files to something like `` `myfileaa' ``, `` `myfileab' ``, and so on, you supply an additional argument that specifies the filename.

The program first sets its defaults, and then tests to make sure there are not too many arguments. It then looks at each argument in turn. The first argument could be a minus followed by a number. If it is, this happens to look like a negative number, so it is made positive, and that is the count of lines. The data file name is skipped over, and the final argument is used as the prefix for the output file names.

```
# split.awk -- do split in awk
# Arnold Robbins, arnold@gnu.ai.mit.edu, Public Domain
# May 1993

# usage: split [-num] [sourcefile] [destinationfile]

BEGIN {
    outfile = "x"    # default
    count = 1000
    if (ARGC > 4)
        usage()

    i = 1
    if (ARGV[i] ~ /^-[0-9]+$/) {
        count = -ARGV[i]
        ARGV[i] = ""
        i++
    }
    # test argv in case reading from stdin instead of file
    if (i in ARGV)
        i++    # skip data file name
    if (i in ARGV) {
        outfile = ARGV[i]
        ARGV[i] = ""
    }

    s1 = s2 = "a"
    out = (outfile s1 s2)
}
```

The next rule does most of the work. `tcount` (temporary count) tracks how many lines have been printed to the output file so far. If it is greater than `count`, it is time to close the current file and start a new one. `s1` and `s2` track the current suffixes for the file name. If they are both `` `z' ``, the file is just too big. Otherwise, `s1` moves to the next letter in the alphabet and `s2` starts over again at `` `a' ``.

```
{
    if (++tcount > count) {
        close(out)
```

```
        if (s2 == "z") {
            if (s1 == "z") {
                printf("split: %s is too large to split\n", \
                        FILENAME) > "/dev/stderr"
                exit 1
            }
            s1 = chr(ord(s1) + 1)
            s2 = "a"
        } else
            s2 = chr(ord(s2) + 1)
        out = (outfile s1 s2)
        tcount = 1
    }
    print > out
}
```

The `usage` function simply prints an error message and exits.

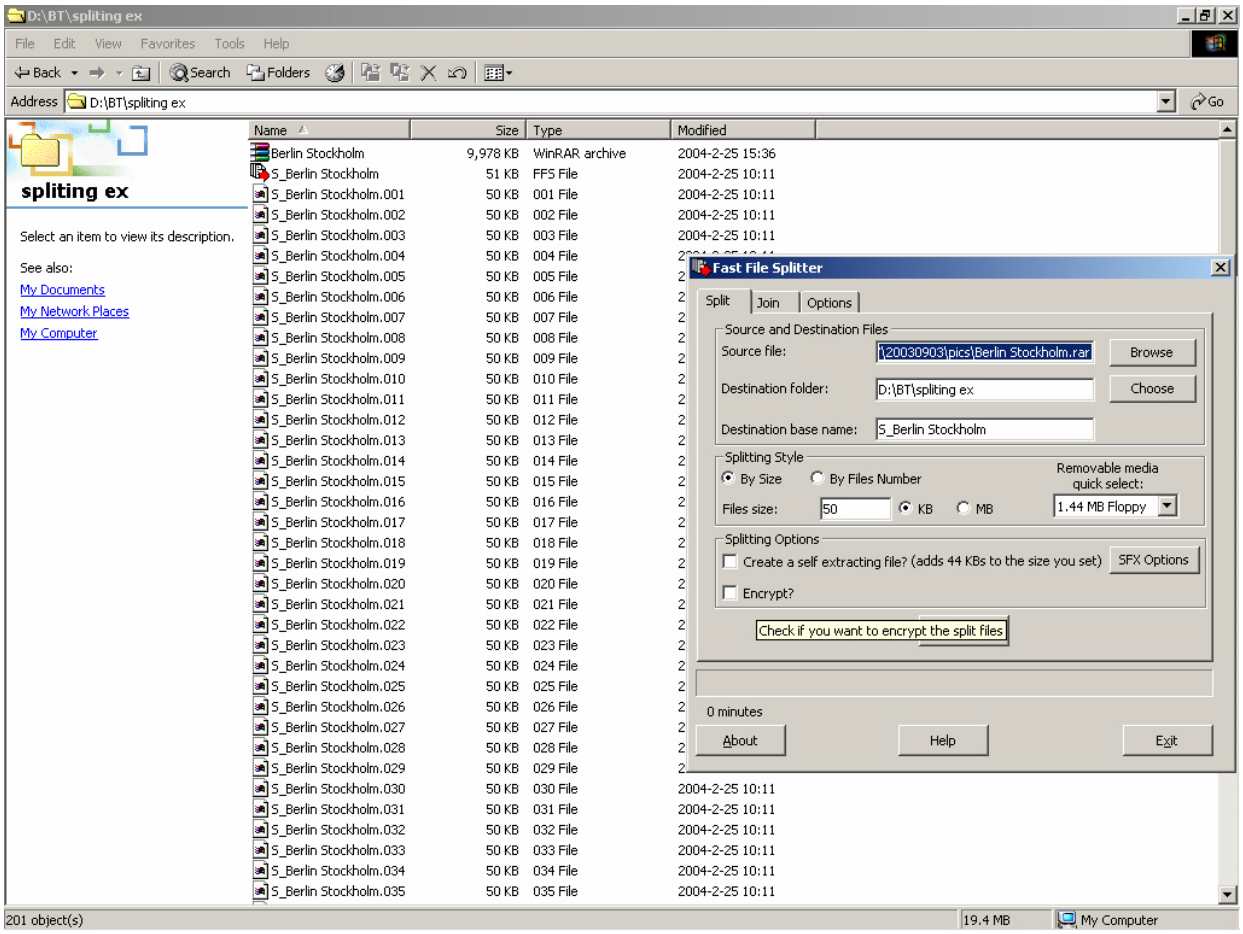
```
function usage(   e)
{
    e = "# usage: split [-num] [sourcefile] [destinationfile]"
    print e > "/dev/stderr"
    exit 1
}
```

The variable `e` is used so that the function fits nicely on the page.

This program is a bit sloppy; it relies on `awk` to close the last file for it automatically, instead of doing it in an `END` rule.

Appendix 2: Snapshoot of using a file splitting tool(freeware)

In this operation, I split a file `Berlin Stockholm.rar` by Fast File Splitter. Then I got 199 blocks of this file and every piece is 50Kb except for the last block with only 28Kb. Use `Join` bottom, they can be restored the original file `Stockholm.rar` in 2-3 seconds.



Appendix 3: Record of a movie file downloaded via BitTorrent

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Documents and Settings\Administrator>ipconfig /all

Windows 2000 IP Configuration

        Host Name . . . . . . . . . . . . : singingbridge
        Primary DNS Suffix  . . . . . . . :
        Node Type . . . . . . . . . . . . : Mixed
        IP Routing Enabled. . . . . . . . : No
        WINS Proxy Enabled. . . . . . . . : No
        DNS Suffix Search List. . . . . . : swipnet.se

Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . : swipnet.se
        Description . . . . . . . . . . . : Realtek RTL8139/810X Family PCI Fast
Ethern
et NIC
        Physical Address. . . . . . . . . : 00-E0-00-99-5F-53
        DHCP Enabled. . . . . . . . . . . : Yes
        Autoconfiguration Enabled . . . . : Yes
        IP Address. . . . . . . . . . . . : 213.100.32.29
        Subnet Mask . . . . . . . . . . . : 255.255.255.192
        Default Gateway . . . . . . . . . : 213.100.32.1
        DHCP Server . . . . . . . . . . . : 130.244.196.150
        DNS Servers . . . . . . . . . . . : 130.244.127.169
        Lease Obtained. . . . . . . . . . : 2004 年 2 月 28 日 10:16:08
        Lease Expires . . . . . . . . . . : 2004 年 2 月 28 日 12:16:08

**********************************host
information************************************


C:\Documents and Settings\Administrator>netstat -a
Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    singingbridge:echo     singingbridge:0        LISTENING
  TCP    singingbridge:discard  singingbridge:0        LISTENING
  TCP    singingbridge:daytime  singingbridge:0        LISTENING
  TCP    singingbridge:qotd     singingbridge:0        LISTENING
  TCP    singingbridge:chargen  singingbridge:0        LISTENING
  TCP    singingbridge:epmap    singingbridge:0        LISTENING
  TCP    singingbridge:1025     singingbridge:0        LISTENING
  TCP    singingbridge:1026     singingbridge:0        LISTENING
  TCP    singingbridge:1039     singingbridge:0        LISTENING
  TCP    singingbridge:1097     singingbridge:0        LISTENING
  TCP    singingbridge:1365     singingbridge:0        LISTENING
  TCP    singingbridge:1370     singingbridge:0        LISTENING
  TCP    singingbridge:1473     singingbridge:0        LISTENING
  TCP    singingbridge:1488     singingbridge:0        LISTENING
  TCP    singingbridge:1491     singingbridge:0        LISTENING
  TCP    singingbridge:1497     singingbridge:0        LISTENING
  TCP    singingbridge:1498     singingbridge:0        LISTENING
  TCP    singingbridge:6000     singingbridge:0        LISTENING
  TCP    singingbridge:6001     singingbridge:0        LISTENING
  TCP    singingbridge:1028     singingbridge:0        LISTENING
  TCP    singingbridge:56666    singingbridge:0        LISTENING
  TCP    singingbridge:netbios-ssn  singingbridge:0        LISTENING
  TCP    singingbridge:1365     220.170.35.125:8882    ESTABLISHED
  TCP    singingbridge:1370     210-85-158-205.cm.apol.com.tw:8883  ESTABLISHED (used
tracert to see number of hops, 20 hops till the last timeout)
  TCP    singingbridge:1473     221.216.102.211:6000   ESTABLISHED
  TCP    singingbridge:1488     221.205.107.136:1883   ESTABLISHED
  TCP    singingbridge:1491     218.80.60.77:6881      ESTABLISHED
  TCP    singingbridge:1497     61.149.22.39:6881      ESTABLISHED
  TCP    singingbridge:1498     218.61.139.44:6000     ESTABLISHED
  TCP    singingbridge:6000     61.191.173.136:2155    TIME_WAIT
  TCP    singingbridge:6000     61.191.173.136:2578    TIME_WAIT
```

```
  TCP    singingbridge:6000     81-223-102-114.Fuenfhaus.Xdsl-line.inode.at:33124
TIME_WAIT
  TCP    singingbridge:6000     81-223-102-114.Fuenfhaus.Xdsl-line.inode.at:33231
TIME_WAIT
  TCP    singingbridge:6000     dsl-210-11-209-190.syd.level10.net.au:3931  TIME_WAIT
(used tracert to see number of hops 22)
  TCP    singingbridge:6000     210.51.228.209:26429   ESTABLISHED
  TCP    singingbridge:6000     211.97.62.204:2369     TIME_WAIT
  TCP    singingbridge:6000     218.11.2.106:13269     TIME_WAIT
  TCP    singingbridge:6000     218.11.2.106:13305     TIME_WAIT
  TCP    singingbridge:6000     218.69.1.10:2044       ESTABLISHED
  TCP    singingbridge:6000     218.246.236.64:65181   ESTABLISHED
  TCP    singingbridge:6001     61.172.29.12:25138     ESTABLISHED (used tracert to
see number of hops, 23 hops)
  TCP    singingbridge:6001     61.185.210.132:44722   ESTABLISHED
  TCP    singingbridge:6001     61.185.237.132:34403   ESTABLISHED
  TCP    singingbridge:6001     61.187.64.210:36890    ESTABLISHED
  TCP    singingbridge:6001     chenpc-216.tamu.edu:4474  ESTABLISHED (used tracert to
see number of hops, 22 hops)
  TCP    singingbridge:6001     202.105.131.102:3086   ESTABLISHED
  TCP    singingbridge:6001     202.105.131.102:3708   TIME_WAIT
  TCP    singingbridge:6001     202.110.201.218:38689  ESTABLISHED
  TCP    singingbridge:6001     dsl-210-11-209-190.syd.level10.net.au:4999
ESTABLISHED
  TCP    singingbridge:6001     210.51.228.209:12572   TIME_WAIT
  TCP    singingbridge:6001     210.51.228.209:21800   TIME_WAIT
  TCP    singingbridge:6001     210.51.228.209:31034   TIME_WAIT
  TCP    singingbridge:6001     210.51.228.209:41882   ESTABLISHED
  TCP    singingbridge:6001     210.51.228.209:50026   TIME_WAIT
  TCP    singingbridge:6001     210.53.6.74:46445      ESTABLISHED (used tracert to
see number of hops)
  TCP    singingbridge:6001     210.211.11.42:4092     ESTABLISHED
  TCP    singingbridge:6001     211.93.112.98:3221     ESTABLISHED
  TCP    singingbridge:6001     211.147.255.124:4758   ESTABLISHED
  TCP    singingbridge:6001     211.158.78.6:1055      ESTABLISHED
  TCP    singingbridge:6001     211.160.16.2:45335     ESTABLISHED
  TCP    singingbridge:6001     211.167.203.34:1499    ESTABLISHED
  TCP    singingbridge:6001     211.167.203.34:3184    TIME_WAIT
  TCP    singingbridge:6001     211.167.203.34:3231    TIME_WAIT
  TCP    singingbridge:6001     218.0.237.158:3009     ESTABLISHED
  TCP    singingbridge:6001     218.7.35.160:3678      ESTABLISHED
  TCP    singingbridge:6001     218.11.2.106:10795     ESTABLISHED
  TCP    singingbridge:6001     218.13.209.176:21977   ESTABLISHED
  TCP    singingbridge:6001     218.66.88.249:3677     ESTABLISHED (used tracert to
see number of hops)
  TCP    singingbridge:6001     218.66.88.249:3788     ESTABLISHED
  TCP    singingbridge:6001     218.66.88.249:3826     ESTABLISHED
  TCP    singingbridge:6001     218.66.210.154:1929    ESTABLISHED
  TCP    singingbridge:6001     218.69.1.10:6781       TIME_WAIT
  TCP    singingbridge:6001     218.72.151.18:3194     ESTABLISHED
  TCP    singingbridge:6001     218.76.145.96:3547     ESTABLISHED
  TCP    singingbridge:6001     218.79.90.178:4761     ESTABLISHED
  TCP    singingbridge:6001     218.79.133.4:4564      ESTABLISHED
  TCP    singingbridge:6001     218.80.60.77:3487      ESTABLISHED
  TCP    singingbridge:6001     218.80.60.77:3488      ESTABLISHED
  TCP    singingbridge:6001     218.80.60.77:4440      TIME_WAIT
  TCP    singingbridge:6001     218.80.60.77:4448      TIME_WAIT
  TCP    singingbridge:6001     218.80.60.77:4727      TIME_WAIT
  TCP    singingbridge:6001     218.86.231.211:3260    ESTABLISHED
  TCP    singingbridge:6001     218.109.32.9:3647      ESTABLISHED
  TCP    singingbridge:6001     219.138.154.173:30083  ESTABLISHED
  TCP    singingbridge:6001     220.186.180.109:4417   ESTABLISHED (used tracert to
see number of hops)
  TCP    singingbridge:6001     220.196.170.3:3909     ESTABLISHED
  TCP    singingbridge:6001     221.204.33.79:3473     ESTABLISHED
  TCP    singingbridge:6001     221.209.150.13:4341    ESTABLISHED (used tracert to
see number of hops)
  UDP    singingbridge:echo     *:*
  UDP    singingbridge:discard  *:*
  UDP    singingbridge:daytime  *:*
  UDP    singingbridge:qotd     *:*
  UDP    singingbridge:chargen  *:*
  UDP    singingbridge:netbios-ns  *:*
  UDP    singingbridge:netbios-dgm  *:*
  UDP    singingbridge:isakmp   *:*
  UDP    singingbridge:router   *:*
  UDP    singingbridge:4500     *:*
```

```
*****************************BitTorrent Peers connected to the
host*****************************


C:\Documents and Settings\Administrator>tracert dsl-210-11-209-190.syd.level10.net.au

Tracing route to dsl-210-11-209-190.syd.level10.net.au [210.11.209.190]
over a maximum of 30 hops:

  1    10 ms   <10 ms   <10 ms  c213-100-32-1.swipnet.se [213.100.32.1]
  2   <10 ms   <10 ms   <10 ms  cty3-core.gigabiteth1-1.swip.net [130.244.189.1]
  3   <10 ms   <10 ms   <10 ms  stb1-core.srp2-0.swip.net [130.244.194.246]
  4   <10 ms   <10 ms   <10 ms  sl-gw10-sto-5-0.sprintlink.net [80.77.97.21]
  5   <10 ms   <10 ms   <10 ms  sl-bb20-sto-8-0.sprintlink.net [80.77.96.37]
  6   <10 ms   <10 ms   <10 ms  sl-bb21-sto-15-0.sprintlink.net [80.77.96.34]
  7   <10 ms   <10 ms    10 ms  sl-bb21-cop-12-0.sprintlink.net [213.206.129.33]
  8    10 ms   <10 ms    10 ms  sl-bb20-cop-15-0.sprintlink.net [80.77.64.33]
  9    80 ms    90 ms    90 ms  sl-bb21-msq-10-0.sprintlink.net [144.232.19.29]
 10    90 ms    90 ms    90 ms  sl-bb22-rly-15-3.sprintlink.net [144.232.19.98]
 11   150 ms   151 ms   160 ms  sl-bb22-sj-10-0.sprintlink.net [144.232.20.186]
 12   150 ms   151 ms   160 ms  sl-bb20-sj-15-0.sprintlink.net [144.232.3.166]
 13   150 ms   161 ms   150 ms  sl-st20-pa-15-1.sprintlink.net [144.232.20.42]
 14   151 ms   150 ms   160 ms  sl-newzeal-1-0.sprintlink.net [144.223.243.18]
 15   300 ms   310 ms   311 ms  p5-0.sjbr1.global-gateway.net.nz [202.37.246.202]
 16   301 ms   310 ms   311 ms  p1-0.sybr3.global-gateway.net.nz [202.50.116.193]
 17   301 ms   310 ms   310 ms  p4-0.sybr2.global-gateway.net.nz [202.50.119.86]
 18   300 ms   311 ms   310 ms  con3.sybr2.global-gateway.net.nz [202.37.246.238]
 19   460 ms   491 ms   310 ms  pos2-0-0.bdr2.hay.connect.com.au [210.8.219.242]
 20   311 ms   300 ms   301 ms  g0-2.cor6.hay.connect.com.au [210.8.134.92]
 21   311 ms   310 ms   311 ms  DLEV140780-5.gw.connect.com.au [210.8.226.141]
 22  1322 ms  2603 ms   721 ms  dsl-210-11-209-190.syd.level10.net.au [210.11.209.190]


Trace complete.



C:\Documents and Settings\Administrator>tracert chenpc-216.tamu.edu

Tracing route to chenpc-216.tamu.edu [165.91.170.208]
over a maximum of 30 hops:

  1   <10 ms   <10 ms   <10 ms  c213-100-32-1.swipnet.se [213.100.32.1]
  2   <10 ms   <10 ms   <10 ms  htg3-core.gigabiteth4-0.swip.net [130.244.189.2]
  3   <10 ms   <10 ms   <10 ms  stb1-core.srp2-0.swip.net [130.244.194.246]
  4   <10 ms   <10 ms   <10 ms  sl-gw10-sto-5-0.sprintlink.net [80.77.97.21]
  5   <10 ms   <10 ms   <10 ms  sl-bb21-sto-8-0.sprintlink.net [80.77.96.41]
  6   <10 ms    10 ms    10 ms  sl-bb21-cop-12-0.sprintlink.net [213.206.129.33]
  7    10 ms    10 ms    10 ms  sl-bb20-cop-15-0.sprintlink.net [80.77.64.33]
  8    10 ms    20 ms    20 ms  sl-bb20-ham-13-0.sprintlink.net [213.206.129.54]
  9    20 ms    21 ms    20 ms  sl-bb21-ams-14-0.sprintlink.net [213.206.129.49]
 10    20 ms    20 ms    20 ms  sl-bb20-ams-15-0.sprintlink.net [217.149.32.33]
 11    40 ms    30 ms    40 ms  213.206.131.46
 12    30 ms    40 ms    40 ms  ae-0-55.mp1.Amsterdam1.Level3.net [213.244.165.97]
 13    30 ms    40 ms    30 ms  so-1-0-0.mp1.London2.Level3.net [212.187.128.49]
 14    90 ms   100 ms   100 ms  so-1-0-0.bbr1.Washington1.Level3.net [212.187.128.138]

 15   121 ms   130 ms   130 ms  unknown.Level3.net [209.247.9.102]
 16   131 ms   130 ms   130 ms  so-6-0.ipcolo1.Dallas1.Level3.net [4.68.112.178]
 17   130 ms   130 ms   140 ms  p0-0.texasamu.bbnplanet.net [4.25.100.2]
 18   131 ms   130 ms   140 ms  csce-7--dmzf-ci-g-10.net.tamu.edu [165.91.254.4]
 19   130 ms   130 ms   140 ms  csce-1.net.tamu.edu [165.91.2.2]
 20   130 ms   130 ms   140 ms  evan-oc22-1.net.tamu.edu [128.194.1.72]
 21     *        *        *     Request timed out.
 22   130 ms   141 ms   130 ms  chenpc-216.tamu.edu [165.91.170.208]

Trace complete.



C:\Documents and Settings\Administrator>tracert 61.172.29.12

Tracing route to 61.172.29.12 over a maximum of 30 hops

  1   <10 ms   <10 ms   <10 ms  c213-100-32-1.swipnet.se [213.100.32.1]
  2   <10 ms   <10 ms   <10 ms  htg3-core.gigabiteth4-0.swip.net [130.244.189.2]
```

```
  3   <10 ms   <10 ms   <10 ms  stb1-core.srp2-0.swip.net [130.244.194.246]
  4   <10 ms   <10 ms    10 ms  sl-gw10-sto-5-0.sprintlink.net [80.77.97.21]
  5   <10 ms   <10 ms   <10 ms  sl-bb21-sto-8-0.sprintlink.net [80.77.96.41]
  6    11 ms    10 ms    10 ms  sl-bb21-cop-12-0.sprintlink.net [213.206.129.33]
  7    10 ms    10 ms    10 ms  sl-bb20-cop-15-0.sprintlink.net [80.77.64.33]
  8    81 ms    90 ms    90 ms  sl-bb21-msq-10-0.sprintlink.net [144.232.19.29]
  9    90 ms    90 ms    90 ms  sl-bb22-rly-15-3.sprintlink.net [144.232.19.98]
 10   150 ms   151 ms   160 ms  sl-bb22-sj-10-0.sprintlink.net [144.232.20.186]
 11   150 ms   151 ms   160 ms  sl-bb24-sj-13-0.sprintlink.net [144.232.3.214]
 12   150 ms   150 ms   151 ms  sl-st21-pa-15-2.sprintlink.net [144.232.9.10]
 13   170 ms   170 ms   171 ms  sl-china4-1-0.sprintlink.net [144.223.243.62]
 14   641 ms   651 ms   641 ms  202.97.51.205
 15   651 ms   671 ms   661 ms  202.97.33.89
 16   651 ms   661 ms   651 ms  202.101.63.253
 17   661 ms   661 ms   661 ms  218.1.1.153
 18   691 ms   691 ms   691 ms  218.1.1.17
 19   671 ms   671 ms   671 ms  218.1.10.162
 20   691 ms   681 ms   691 ms  218.1.10.182
 21     *        *        *     Request timed out.
 22     *        *        *     Request timed out.
 23   681 ms   640 ms   661 ms  61.172.29.12

Trace complete.
```

IMIT/LCN 2004-05

www.kth.se